Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Contribuciones al desarrollo de plataforma estandarizada para la simulación de sistemas de automatización

Autor: Juan Gómez Jiménez
Tutor: Juan Manuel Escaño González
Cotutor: Adolfo J. Sánchez del Pozo Fernández

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Contribuciones al desarrollo de plataforma estandarizada para la simulación de sistemas de automatización

Autor:
Juan Gómez Jiménez

Tutores:
Juan Manuel Escaño González
Adolfo J. Sánchez del Pozo Fernández

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Contribuciones al desarrollo de plataforma estandarizada para la simulación de sistemas de automatización

Autor: Juan Gómez Jiménez

Tutores: Juan Manuel Escaño González y Adolfo J. Sánchez del Pozo Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia, que me ha estado apoyando en todo momento, en los momentos buenos y en los malos

A mis maestros, a Juanma y Adolfo, por ayudarme a sacar adelante este proyecto

ÍNDICE

Índice viii

Índice de figuras x

1 Resumen 1

2 Introducción 2
2.1. Estado del arte 2
2.2. Motivación 3
2.3. Objetivo 3
2.4. Organización del proyecto 3

3 Unity 3D 5
3.1. Descarga e instalación de Unity 3D 5
3.2. Instalando Unity en Windows desde línea de comandos 5
3.3. Instalación del Editor de Unity 6
3.4. Instalación del Web Player 6
3.5. Instalación de Assets Estándar 6
3.6. Instalación de Proyecto Ejemplo 6
3.7. Instalando Unity en OS X desde la línea de comando 6
3.8. Instalación del Editor de Unity 6
3.9. Instalación del Web Player 7
3.10. Instalación de Assets Estándar 7
3.11. Instalación de Proyecto Ejemplo 7
3.12. Interfaz de Unity 7

4 Estructura de la aplicación 10
4.1. Lenguaje de programación: C# 10
4.2. Creación de un nuevo objeto y asignación de comportamiento 10

5 Descripción detallada de los elementos físicos 13
5.1. Cinta transportadora 14
5.2. Mesa giratoria 16
5.3. Grúa pórtico 19
5.3.1. Desplazamiento trasversal 20
5.3.2. Desplazamiento vertical 21
5.3.3. Apertura y cierre de pinzas 22
5.3.4. Sensores detectores de caja 23
5.3.5. Sensores de apertura y cierre de pinzas 24
5.3.6. Programa asociado a las pinzas 25
5.4. Sensores láser 31
5.5. Pistones de encuadre de cajas 36
5.6. Creador de cajas 39
5.7. Destructor de cajas 41

6 Descripción detallada de los elementos visuales 3D 42
6.1. Cinta transportadora 43
6.1.1. Soporte central 43
6.1.2. Patas de la cinta 45
6.1.3. Cinta de transporte 47
6.2. Mesa giratoria 50
6.2.1. Base 50
6.2.2. Parte móvil 51
6.3. Grúa pórtico 54
6.3.1. Estructura base 54

6.3.2. Pieza móvil 56
6.3.3. Brazo extensible 57
6.3.4. Soporte y pinzas 58
6.3.5. Animaciones 59
6.4. Pistón *62*
6.5. Sensor láser *64*

7 Descripción de la planta industrial diseñada **65**
7.1. Resumen de elementos de la planta *66*
7.2. Coordinación de todos los elementos. Explicación del script Control. *67*

8 Conclusiones y trabajos futuros **68**

9 Bibliografía **69**

ÍNDICE DE FIGURAS

Figura 1. Componentes a instalar por defecto en Unity 5
Figura 2. Partes de la interfaz propia de Unity 7
Figura 3. Interfaz de la ventana del explorador 8
Figura 4. Interfaz de la ventana del inspector 8
Figura 5. Interfaz de la ventana de jerarquía 9
Figura 6. Interfaz de la ventana de escena 9
Figura 7. Interfaz de la ventana de juego 9
Figura 8. Creación de un objeto 3D 10
Figura 9. Ejes de traslación 11
Figura 10. Ejes de rotación 11
Figura 11. Ejes de modificación de escala 11
Figura 12. Ubicación del nuevo material en la ventana de explorador 11
Figura 13. Especificaciones de material en el inspector 11
Figura 14. Creación de nuevo script 12
Figura 15. Desactivación de la textura del objeto para evitar su visibilidad 13
Figura 16. Visión general de la cinta transportadora 14
Figura 17. Elementos físico y visual de la cinta 14
Figura 18. Esquema de movimiento de la cinta 15
Figura 19. Rutina de giro de la mesa 16
Figura 20. Elementos físicos y sensores de la mesa giratoria 18
Figura 21. Activación de la configuración de trigger en un objeto 19
Figura 22. Visión general de la grúa pórtico 19
Figura 23. Desplazamiento horizontal de la grúa 20
Figura 24. Diagrama de estados de la rutina de movimiento horizontal de la grúa programado en el animator controller 21
Figura 25. Desplazamiento vertical de la grúa 21
Figura 26. Diagrama de estados de la rutina de movimiento horizontal de la grúa programado en el animator controller 22
Figura 27. Modificación de la velocidad de la animación 22
Figura 28. Apertura y cierre de pinzas de la grúa 23
Figura 29. Elementos físicos de la grúa para detectar la caja 23
Figura 30. Sensores de apertura y cierre de pinzas de la grúa 24
Figura 31. La caja deja de ser hija de la grúa en el momento en que las pinzas se abren al soltarla en la posición final 29
Figura 32. Esquema del modelo reflectivo del sensor láser 31
Figura 33. Esquema del modelo de barrera del sensor láser 32

Figura 34. Esquema del modelo retroreflectivo del sensor láser 32
Figura 35. Visión general del sensor láser del proyecto 32
Figura 36. Configuración de material luminescente 33
Figura 37. Panel de información de estado de los sensores 34
Figura 38. Activación del sensor 0 por parte de una caja 34
Figura 39. Indicadores de sensor encendido y apagado creados y almacenados en la carpeta Resources ... 35
Figura 40. Visión general de los pistones de encuadre 36
Figura 41. Un sensor de detección de cajas del pistón, ocupando toda la superficie del mismo 37
Figura 42. Los pistones se detienen antes de cuadrar completamente toda la caja debido a la existencia de un único sensor por pistón 37
Figura 43. Dos sensores de detección de cajas del pistón, ocupando cada uno la mitad de la superficie del pistón 38
Figura 44. La caja queda completamente cuadrada al ubicar dos sensores en cada pistón 38
Figura 45. Aparición de nueva caja por el túnel de entrada de la planta 39
Figura 46. Visión del sensor Habilitadornuevacaja situado en el túnel de entrada 39
Figura 47. Elemento destructor de cajas situado al final del recorrido de la planta 41
Figura 48. Apariencia final de la cinta transportadora 43
Figura 49. Silueta del perfil del soporte central de la cinta 43
Figura 50. Uso de la herramienta Extrusión 44
Figura 51. Soporte central de la cinta tras haberle aplicado volumen 44
Figura 52. Adición de tapas laterales para redondear el filo y captar mejor la luz 44
Figura 53. Detalle del canto redondeado del soporte central 45
Figura 54. Plataforma de la pata de la cinta 45
Figura 55. Función unión de splines en Cinema4D 46
Figura 56. Perfil de la barra proveniente de la pata de la cinta 46
Figura 57. Unión de la plataforma y la barra para formar la pata completa de la cinta 46
Figura 58. Perfil de la guía de la cadena 47
Figura 59. Dimensiones y configuración de la guía 47
Figura 60. Bloque componente de la cadena de la cinta 47
Figura 61. Dimensiones de cada bloque componente de la cadena 48
Figura 62. Herramienta clonador de Cinema4D 48
Figura 63. Configuración de la animación de la cinta 49
Figura 64. Ubicación del cubo dentro del objeto clonador para aplicar dicha herramienta 49
Figura 65. Vista general de la cadena de la cinta 49
Figura 66. Comparación de la cinta creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color 50
Figura 67. Apariencia final de la mesaq giratoria 50
Figura 68. Pieza base de la mesa 50
Figura 69. Perfil de la base de la mesa 51

Figura 70. Configuración de redondeo de esquinas del perfil de la base de la mesa.....51
Figura 71. Parte móvil de la mesa.....52
Figura 72. Cilindro elevador de la mesa.....52
Figura 73. Plataforma base y cuatro soportes de la cinta.....52
Figura 74. Soporte central de la cinta de la mesa giratoria.....53
Figura 75. Cadena de la cinta de la mesa giratoria.....53
Figura 76. Comparación de la mesa giratoria creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color.....53
Figura 77. Apariencia final de la grúa.....54
Figura 78. Apariencia final de la estructura base de la grúa.....54
Figura 79. Perfil de una pata de la estructura (a) y aplicación de volumen al mismo (b).....54
Figura 80. Herramienta Convertir a editable, para hacer el conjunto objeto-extrusión un único objeto más manejable.....55
Figura 81. Herramienta Simetría de Cinema4D.....55
Figura 82. Colocación del objeto dentro del objeto Simetría para aplicar el efecto de la herramienta sobre el mismo.....55
Figura 83. Duplicación de forma simétrica de una pata de la estructura de la grúa.....56
Figura 84. Estructura finalizada.....56
Figura 85. Pieza móvil de la grúa.....57
Figura 86. Perfil de la pieza móvil.....57
Figura 87. Brazo extensible de la grúa.....57
Figura 88. Dimensiones del anillo que forma el perfil del cilindro exterior.....58
Figura 89. Pieza soporte de pinzas de la grúa.....58
Figura 90. Diseño de la mitad del perfil de la pieza.....59
Figura 91. Obtención de la pieza completa mediante el uso de la herramienta Simetría.....59
Figura 92. Línea de tiempo de Cinema4D.....60
Figura 93. Interfaz de barra de herramientras de la seciencia.....61
Figura 94. Comparación de la grúa creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color.....62
Figura 95. Vista general del piston de encuadre.....62
Figura 96. Parte móvil del pistón.....62
Figura 97. Perfil del soporte y soporte con volumen aplicado.....63
Figura 98. Comparación del pistón creado en Cinema4D y el mismo importado en Unity tras aplicarle iluminación, texturas y corrección de color.....63
Figura 99. Vista general del sensor láser.....64
Figura 100. Comparación del sensor láser creado en Cinema4D y el mismo importado en Unity tras aplicarle iluminación, texturas y corrección de color.....64
Figura 101. Esquema general del flujo de movimiento de la planta.....65
Figura 102. Vista general de la planta con corrección de color aplicada.....66

1 RESUMEN

El trabajo que se desarrolla a continuación trata exponer el desarrollo de un software de simulación de plantas industriales con capacidad de conexión con controladores industriales, permitiendo al programador la interacción con elementos de simulación de alto grado de interactividad y realismo.

Para el desarrollo del mismo, se ha hecho uso de Unity 3D, uno de los programas más usados para el diseño y desarrollo de videojuegos, con la ventaja de que es de uso libre.

El simulador al que se ha llegado para este trabajo expone un conjunto de elementos industriales determinados (cintas transportadoras, mesas de giro, pórtico) con una distribución a modo de ejemplo, sin procurar buscar una finalidad concreta, ya que, el objetivo final de dicho simulador es que sea una plataforma abierta editable, capaz de ser configurada por cada usuario.

2 INTRODUCCIÓN

Los motores gráficos son entornos de trabajo diseñados específicamente para la creación y desarrollo de videojuegos. Al principio, la tecnología software que se empleaba para el desarrollo de videojuegos no diferenciaba de forma clara los distintos componentes empleados, resultando una programación muy homogénea y poco estructurada. A mediados de los 90 se desarrolló el juego conocido como Doom, el cual mostraba una arquitectura que permitía la reutilización de muchos componentes. A partir de ese momento se empezó a acuñar el término "Game-Engine" para referirse a la utilización de ciertos componentes básicos y comunes sobre los cuales se basa el resto de la programación. Tal ha sido su evolución y optimización que los motores gráficos están suponiendo un avance en el campo del entretenimiento puro a nuevas aplicaciones enfocadas a otros entornos o sectores, como pueden ser la formación y entrenamiento a través del desarrollo de simuladores, denominado Serious Games, u otro tipo de aplicaciones de Realidad Virtual enfocadas al sector industrial. [1]

De esta forma, los entornos virtuales sirven como herramienta previa para verificar el buen comportamiento de la instalación o, por el contrario, para detectar errores sin afectar a los componentes reales. Por tanto, el comportamiento de los elementos en el simulador debe ser lo más fiel posible a la realidad, contemplando fuerzas como la gravedad, fricción, inercia, etc.

La simulación de dichas fuerzas que surgen en los objetos físicos de la instalación se realiza a través de lo que se conoce como motor físico, sistema de software que implementa una simulación aproximada de ciertos sistemas físicos, como dinámica de sólidos rígidos (incluyendo detección de colisiones), dinámica de cuerpos deformables, dinámica de fluidos, etc, en tiempo real.

El motor de físicas es una parte fundamental de un motor de videojuegos. Su implementación no es nada sencilla, y de hecho es actualmente un campo de investigación en el que poco a poco ir mejorando en fidelidad al comportamiento real de los objetos. Debido a esta complejidad, pocas compañías implementan su motor de físicas, sino que usan el que proporciona una compañía especializada en la implementación de motores de físicas (middleware), para incorporarlo a su motor, o al motor de un tercero.

El motor de físicas más famoso y usado en la industria es Havok Physics, de la empresa Havok. [2]

2.1. Estado del arte

A la hora de hablar de entornos de simulación 3D de plantas industriales, existen varias propuestas. Sin embargo, a nivel educacional, uno de los más conocido es "Factory IO", que presenta más de 80 componentes industriales y 20 escenarios distintos, brindando gran versatilidad a la hora de editar nuevas plantas, así como un gran realismo, tanto gráfico como respecto a la dinámica de los objetos (fuerzas de gravedad, inercias, etc).

Además, cuenta con drivers para interaccionar con PLC, SoftPLC, Modbus y muchas otras tecnologías.

El principal problema de este software es que, tras una prueba gratuita de 30 días, requiere el pago de una cuota anual elevada, inadmisible para muchos estudiantes.

Por otro lado, en diciembre de 2012 se publicó un estudio [3] en el que se desarrolla una plataforma de simulación 3D en código abierto (*open source*) de plantas industriales, con su propio motor físico, y un diseño de numerosos elementos industriales comunes, como cintas transportadoras, mesas de giro, sensores de presencia, pistones, un pórtico tipo grúa, y hasta un brazo robótico de tres grados de libertad. Además, implementó un editor de planta para añadir y ubicar elementos de acuerdo con las especificaciones del usuario. El problema de este proyecto es que ha quedado obsoleto por las herramientas que se utilizaron. Durante el desarrollo del mismo, se empleó *Microsoft XNA*, cuya última versión estable data de septiembre del 2010, por lo que el proyecto abarcado por *Microsoft* está actualmente sin soporte (aunque existe un proyecto de *software* libre).

En septiembre de 2018 se realizó una continuación a este estudio [4], tratando de sustituir el software por uno más actual, eligiendo el programa denominado Unity 3d, uno de los más conocidos en el ámbito de desarrollo de videojuegos, otorgando un gran soporte a las librerías necesarias para el desarrollo de dicho trabajo. En dicho proyecto se diseñó una estructura sencilla, compuesta por cintas transportadoras, sensores de presencia, spawners (elementos de creación de nuevas cajas) y destructores (elementos de destrucción de cajas al finalizar el recorrido de la planta). El spawner creaba cajas cada cierto tiempo, seleccionando de forma aleatoria entre 4 tamaños diferentes de cajas, con intención de mostrar el funcionamiento de la planta con cajas de dimensiones variadas. Para terminar, se realizó un pequeño experimento de comunicación por protocolo MODBUS, conectando el software desarrollado en Unity a un simulador de PLC, denominado SoMachine Basic, software que recrea el comportamiento de un PLC convencional.

2.2. Motivación

El principal incentivo de este proyecto ha sido el hecho de desarrollar una plataforma de simulación de plantas industriales que trate de minimizar las limitaciones que presentan los simuladores comerciales.

En primer lugar, diseñar un entorno de uso gratuito, al que tenga acceso todo el mundo.

Por otro lado, que sea un software de **código abierto (open source)**, de tal forma que nuevos usuarios puedan aportar nuevos elementos y creaciones, contribuyendo al desarrollo del software. Para conseguir esto, será necesario definir una estructura de programación para evitar problemas de incompatibilidad garantizar un correcto funcionamiento.

Otro gran incentivo ha sido su uso en el ámbito de la enseñanza, permitiendo el acceso a universidades que puedan contribuir también en el desarrollo de la aplicación.

2.3. Objetivo

En este documento se presenta el desarrollo de una plataforma de simulación 3D de plantas industriales.

El objetivo es la creación de un simulador de planta industrial avanzado, compuesta por elementos frecuentemente usados (cintas transportadoras, mesas de giro, pórticos, pistones, sensores…) que sea funcional, es decir, que todos los elementos estén conectados de forma que funcionen de forma coordinada y automática, que refleje el comportamiento que tendría la planta real, permitiendo así un uso para la investigación, enseñanza, o diseños de plantas reales.

2.4. Organización del proyecto

El documento está organizado en varias secciones.

En primer lugar, se propone una introducción a Unity 3D, el software empleado para el desarrollo del proyecto. Se explicará la instalación y el uso del programa, para poner más fácil al usuario su uso.

A continuación, se presenta una exposición de la estructura del proyecto. En esta parte se mostrarán otros aspectos como el lenguaje de programación y un resumen del funcionamiento de la aplicación, haciendo uso de diagramas de flujo para facilitar la comprensión.

Por consiguiente, se muestra el desarrollo de los elementos físicos. Se detallarán, uno a uno, cada uno de los elementos creados, explicando detalladamente su modelado y funcionamiento.

Después de los elementos físicos, se explica el diseño de la parte visual de cada elemento paso a paso.

Luego se expone una descripción de la planta como entidad, y se explica el desarrollo del proceso de coordinación de todos los elementos.

Finalmente se relatan una serie de conclusiones y trabajos futuros. En este apartado se desarrollan ideas acerca de futuras posibles aportaciones al proyecto, como por ejemplo la comunicación del simulador con un PLC convencional, o la creación de un editor de escenarios que permita añadir, quitar y distribuir elementos en una planta conforme decida el usuario.

3 UNITY 3D

Durante la planificación del proyecto, la elección del software es de vital importancia, ya que es la plataforma en la que se basará todo el desarrollo, y debido a que la versatilidad es uno de los objetivos a alcanzar, es necesario que dicho software sea comúnmente conocido y goce de amplias librerías para tener el mínimo número de restricciones posibles, así como facilitar el aprendizaje de su uso debido a la extensa documentación existente en internet.

3.1. Descarga e instalación de Unity 3D

A continuación, se detalla el proceso de descarga e instalación de Unity 3D paso a paso. [5]

La descarga del programa puede hacerse de forma gratuita en la página principal de Unity.

El instalador utiliza un Asistente de Descarga y tiene instrucciones detalladas paso a paso.

Desde la versión 5.0 de Unity en adelante, el Asistente de Descarga de Unity, un pequeño ejecutable (de aproximadamente 1 MB de tamaño), permite seleccionar qué componentes del editor Unity se desean descargar e instalar. Si no se está seguro sobre qué componentes se desean instalar, deje las selecciones predeterminadas y oprima **Continue**, siguiendo las instrucciones del instalador.

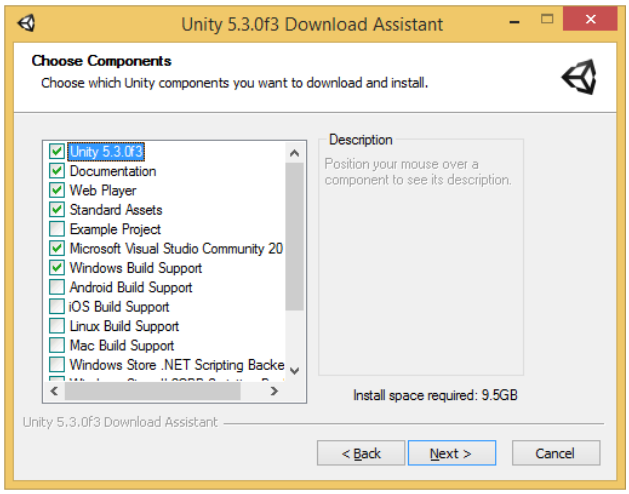

Figura 1. Componentes a instalar por defecto en Unity

Existe otra opción para descargar e instalar todos los componentes por separado, sin utilizar el Asistente de Descarga. Los componentes son programas ejecutables y paquetes instaladores normales, con lo que puede resultar más sencillo, especialmente si es un nuevo usuario de Unity, emplear el Asistente de Descarga. Algunos usuarios, como aquellos que desean automatizar el despliegue de Unity en una organización, pueden preferir una instalación desde línea de comando.

3.2. Instalando Unity en Windows desde línea de comandos

Las siguientes opciones se pueden utilizar cuando se está instalando el Editor de Unity y otros componentes desde la línea de comandos de Windows. Es conveniente tener en cuenta que los argumentos del instalador por línea de comandos distinguen entre mayúsculas y minúsculas.

3.3. Instalación del Editor de Unity

/S	Realiza una instalación silenciosa (sin preguntas).
/D=PATH	Establece el directorio de instalación predeterminado. Útil cuando se combina con la opción de instalación silenciosa. El directorio por defecto es `C:\Program Files (x86)\Unity\` (32-bit) o `C:\Program Files\Unity\` (64-bit)

Ejemplo:

```
UnitySetup64.exe /S /D=E:\Development\Unity
```

En este caso, se instalará Unity en modo silencioso en el directorio E:\Development\Unity, que será la raíz de la instalación de Unity. El ejecutable del Editor de Unity se ubicará en E:\Development\Unity\Editor\Unity.exe. El argumento "/D" tiene que ser el último y sin comillas, incluso si la ruta contiene espacios.

3.4. Instalación del Web Player

Instalación silenciosa del Web Player (No se puede especificar el directorio de instalación):

```
UnityWebPlayer.exe /S
```

3.5. Instalación de Assets Estándar

De manera silenciosa instala los Assets Estándar. Si se especifica la carpeta, utilice la carpeta "raíz" de Unity, p.e. la carpeta que contiene la carpeta Editor y no dónde Unity.exe está instalado.

```
UnityStandardAssetsSetup.exe /S /D=E:\Development\Unity
```

3.6. Instalación de Proyecto Ejemplo

De manera silenciosa se instala el Proyecto Ejemplo. La carpeta por defecto es C:\Users\Public\Documentation\Unity Projects\Standard Assets Example Project

```
UnityExampleProjectSetup.exe /S /D=E:\Development\Unity
```

3.7. Instalando Unity en OS X desde la línea de comando

Los instaladores individuales de Unity son proporcionados como archivos .pkg, los cuales pueden ser instalados utilizando `installer` como se describe abajo.

3.8. Instalación del Editor de Unity

Instala la carpeta `/Applications/Unity` en el volumen destino especificado:

```
sudo installer [-dumplog] -package Unity.pkg -target /
```

3.9. Instalación del Web Player

El Reproductor Web de Unity está instalado en `/Library/Internet Plug-ins/Unity Web Player.plugin` en Mac.

```
sudo installer [-dumplog] -package WebPlayer.pkg -target /
```

3.10. Instalación de Assets Estándar

Instala a la carpeta `/Applications/Unity/Standard Assets` en el volumen especificado:

```
sudo installer [-dumplog] -package StandardAssets.pkg -target /
```

3.11. Instalación de Proyecto Ejemplo

Instala a la carpeta `/Users/Shared/Unity/Standard-Assets` en el volumen especificado:

```
sudo installer [-dumplog] -package Examples.pkg -target /
```

Una vez descargado e instalado, a continuación, se presenta la interfaz del programa, para poder conocer un poco el entorno de trabajo.

3.12. Interfaz de Unity

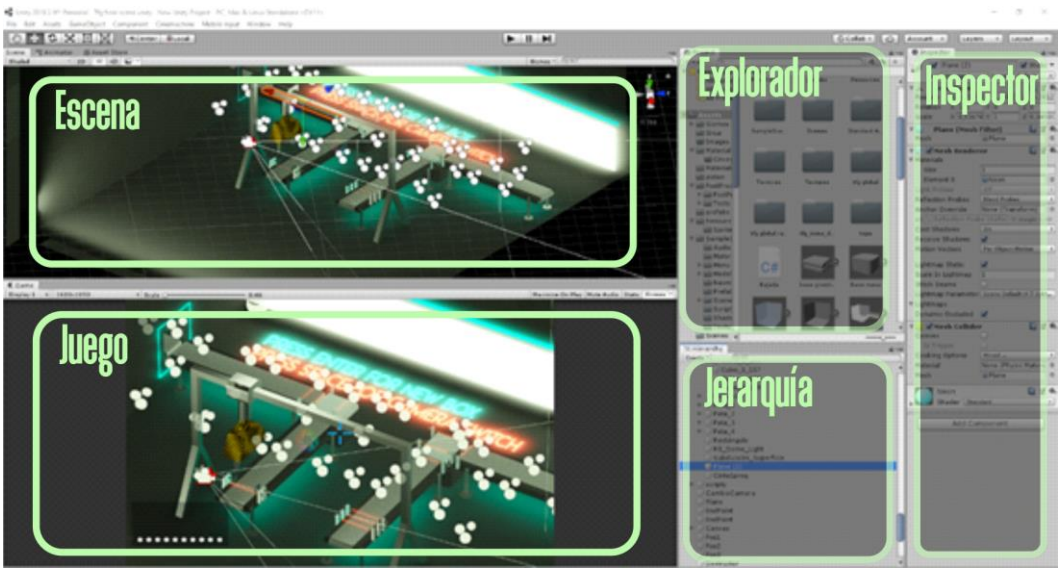

Figura 2. Partes de la interfaz propia de Unity

El editor de Unity 3D es uno de los más sencillos y potentes del mercado. Su interfaz se divide en 5 vistas principales:

1. **Explorador**: Lista todos los elementos (o activos) del proyecto. Permite ordenar de forma sencilla la aplicación. En esta vista se encuentran las imágenes, escenas, scripts, audios, prefabs (objetos con valores predeterminados por el usuario), texturas, atlas y todos los elementos usados en el juego o aplicación.

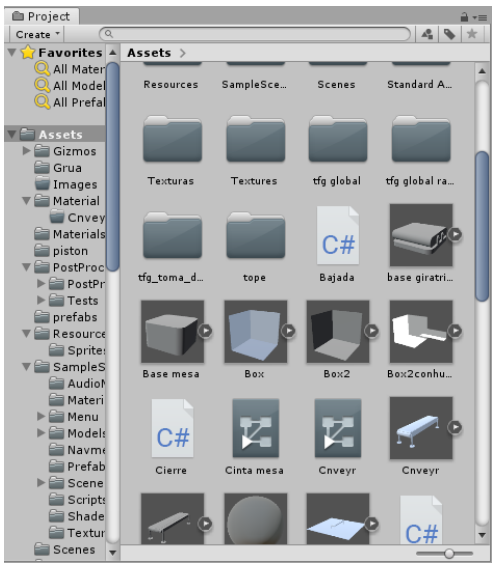

Figura 3. Interfaz de la ventana del explorador

2. **Inspector**: Muestra y define las propiedades de los elementos del proyecto. Modifica valores de forma rápida, cambia texturas arrastrando ficheros desde el Explorador, añade scripts, guarda prefabs…

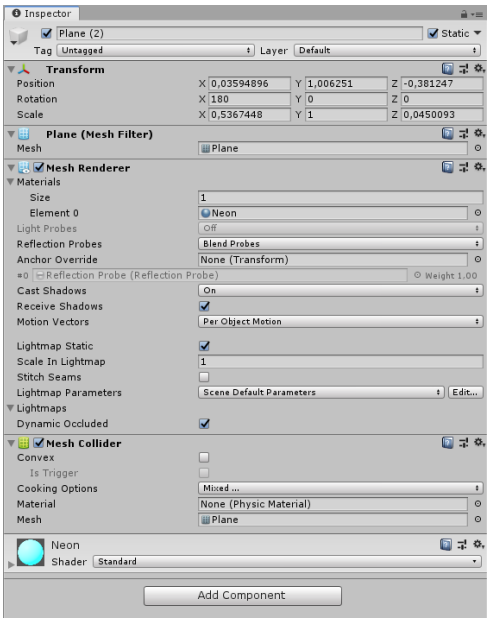

Figura 4. Interfaz de la ventana del inspector

3. **Jerarquía**: Lista en la cual se muestran cada uno de los elementos de la escena de forma rápida y visual. Al seleccionar un objeto, se despliega un nuevo grupo que contiene todos los elementos por los que está formado dicho objeto.

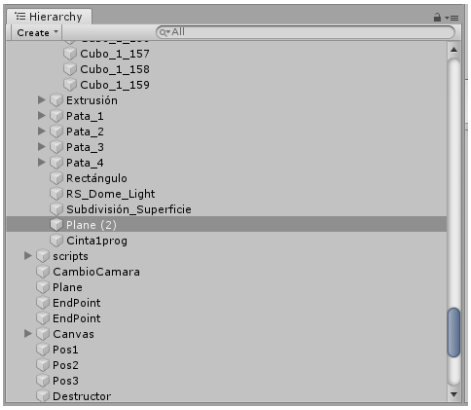

Figura 5. Interfaz de la ventana de jerarquía

4. **Escena**: Diseño y maqueta del juego completo o una pantalla o sección de éste. Cada escena representa un nivel o sección diferente del juego (portada, nivel 1, nivel 2, login…). Esta es la pantalla usada por el usuario puede moverse y orientarse por el entorno, para interactuar con cada uno de los objetos, y ubicarlos conforme a sus necesidades. En esta interfaz también se encuentran ubicados otros elementos claves, como son la fuente de luz que ilumina la escena y la cámara principal, cámara que define la vista que tendrá la aplicación de cara al usuario.

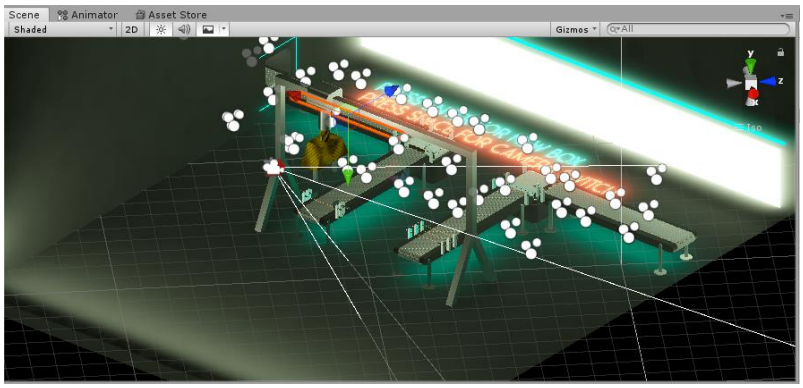

Figura 6. Interfaz de la ventana de escena

5. **Juego**: Pantalla que muestra la vista del juego o aplicación durante su ejecución. Dicha vista depende de la posición y orientación que se le de a la cámara, objeto que puede seleccionarse y moverse en la pantalla de escena, ya mencionado anteriormente.

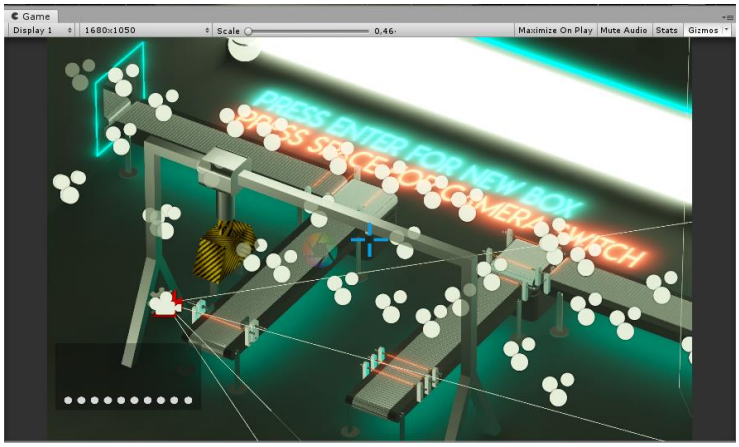

Figura 7. Interfaz de la ventana de juego

4 ESTRUCTURA DE LA APLICACIÓN

4.1. Lenguaje de programación: C#

C# (leído en inglés "C Sharp" y en español "C Almohadilla") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

C# es el único lenguaje de programación que ha sido diseñado específicamente para ser utilizado en la plataforma .NET, por lo que programar en dicha plataforma lo hace mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes, debido a su carencia de elementos heredados innecesarios. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET

Contiene una estructura de programación muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y hacer que su aprendizaje a los desarrolladores sea un proceso sencillo.

Por tanto, a modo de resumen, se puede decir que C++ es un lenguaje que trata de quedarse con los puntos positivos y ventajas de otros lenguajes existentes, tales como Visual Basic, Java o C++ y las combina en uno solo. [6]

4.2. Creación de un nuevo objeto y asignación de comportamiento

A la hora de añadir nuevos objetos a un proyecto de Unity, existen dos facetas principales: la parte visual del objeto, es decir, la apariencia del mismo (color, forma, textura, material, luminosidad…), y la parte que se relaciona con el comportamiento del objeto.

Para crear un nuevo objeto, seleccionar el botón *Create* situado en la parte superior de la ventana de jerarquía, donde se desplegarán todos los tipos de objetos que podemos crear, entre los cuales existe la posibilidad de crear objetos 3D y 2D. Dentro de la pestaña *3D objects* se encuentran todas las figuras básicas que se pueden elegir para dicho objeto. Para modelar formas más sofisticadas será necesario acudir a un programa de modelado 3D, como el Cinema4D, utilizado para la realización de las formas de los objeto de este proyecto, y que se presentará más adelante.

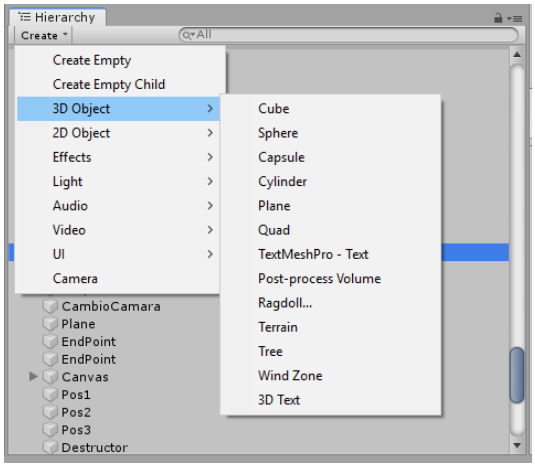

Figura 8. Creación de un objeto 3D

Una vez seleccionada la forma para el objeto, éste aparecerá en el centro de la pantalla. La posición, orientación y escala pueden ser modificadas en la ventana *Inspector*, o usando las flechas que se muestran encima del objeto.

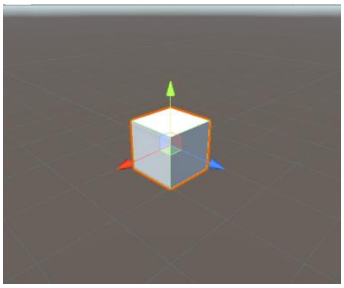

Figura 9. Ejes de traslación

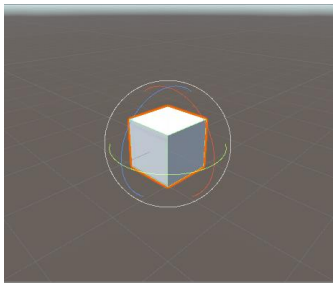

Figura 10. Ejes de rotación

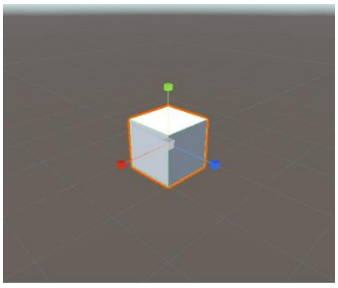

Figura 11. Ejes de modificación de escala

Para modificar la apariencia del objeto, es necesario añadir un nuevo material, y asignarlo a dicho objeto. Para ello, en la ventana del explorador se selecciona *Create > Material.*

Una vez creado, aparecerá con un icono de una esfera en la ventana del explorador. Si se selecciona dicho material, en el inspector aparecen todas las propiedades a definir por el usuario: color, brillo, luminancia, rugosidad…

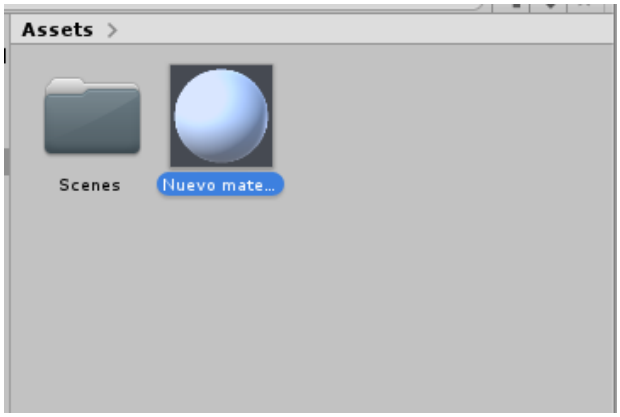

Figura 12. Ubicación del nuevo material en la ventana de explorador

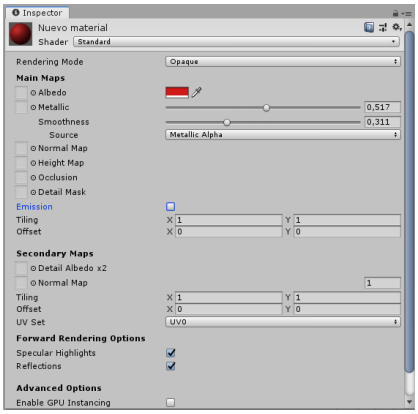

Figura 13. Especificaciones de material en el inspector

Finalmente, para aplicar el material a un objeto basta con mantener el ratón presionado sobre el material, y arrastrarlo sobre dicho objeto, que tomará la apariencia del material creado.

Una vez creado el objeto, el siguiente paso es añadirle un comportamiento determinado. Dependiendo del objeto que sea y de la función que desempeñe en el juego o en la aplicación tendrá un comportamiento determinado: un muro que se mueve horizontalmente, una puerta que se abre al presionar un botón, una plataforma que sube y baja, el personaje principal que se desplaza y orienta según las flechas del teclado, etc. Realmente aquí las posibilidades son infinitas. El único requisito es saber programar dicho comportamiento.

La programación en Unity se realiza en su mayor medida en C#, un lenguaje de programación muy usado y conocido. Por tanto, para crear un nuevo comportamiento y asignarlo a un objeto debemos crear un nuevo script, una hoja de texto donde se escribe el código, que luego se puede ejecutar y añadir al objeto deseado.

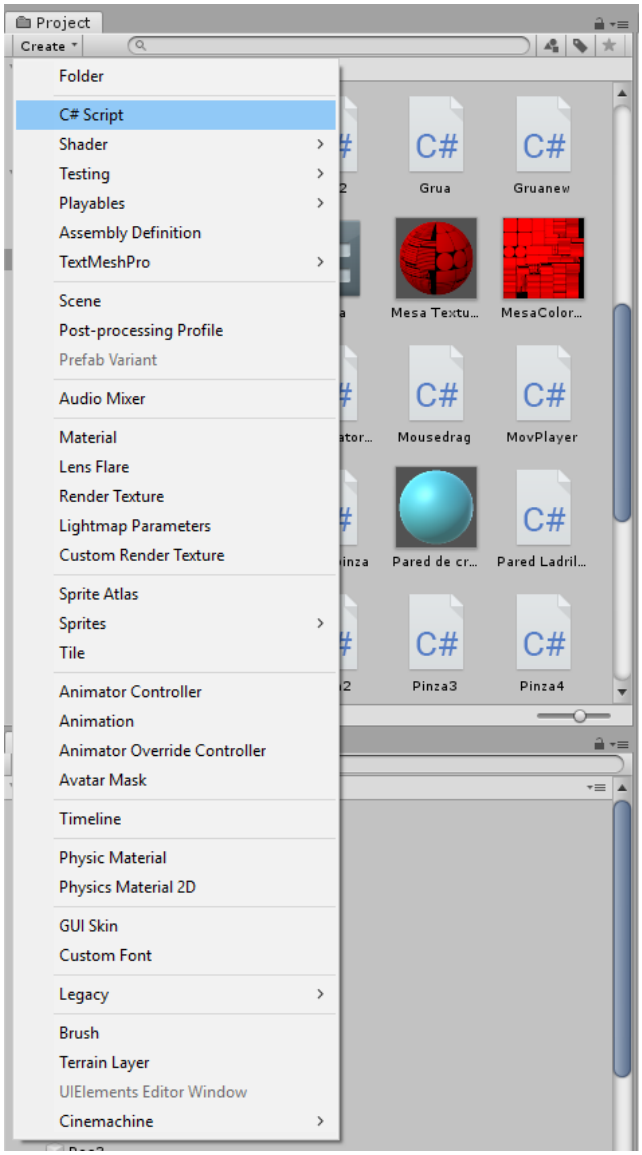

Figura 14. Creación de nuevo script

5 DESCRIPCIÓN DETALLADA DE LOS ELEMENTOS FÍSICOS

En este proyecto de han creado una serie de elementos comúnmente hallados en una planta industrial, a modo de un primer acercamiento a un simulador. Dichos elementos constan de dos partes principales bien diferenciadas: una superficie que contiene la programación del comportamiento del objeto, denominada a partir de ahora como *estructura*, y otra parte que conforma la parte visible del objeto, que será denominada *textura*.

- MODELO FISICO:

La *estructura* del objeto consiste en una o varias superficies de grosor mínimo y dimensiones iguales a las dimensiones de la/s superficie/s del objeto que harían contacto con la caja transportada. Para hacer más sencilla la explicación, se tomará una superficie única, aunque luego se indicará que hay objetos que contienen más de una.

Esta superficie contiene la programación necesaria que define el comportamiento del objeto, ya que es la única parte del objeto que interactúa con el cuerpo dinámico.

Sin embargo, para evitar que sea visible por el usuario, se desactiva la opción *Mesh Renderer*, que es la textura de esta superficie, y así queda transparente.

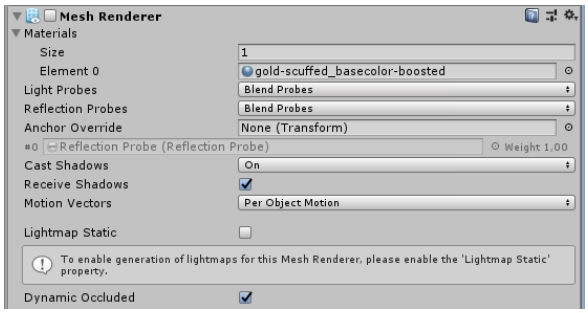

Figura 15. Desactivación de la textura del objeto para evitar su visibilidad

- MODELO 3D O VISUAL

La *textura* es la parte visual del objeto, que trata de asemejar el objeto lo más posible al elemento industrial real. En este proyecto, se ha separado la textura de la estructura para separar problemas y trabajar de forma más sencilla. Por tanto, todo lo que pertenezca a la textura de un elemento no tendrá una programación asignada.

A la hora de modelar la forma de un objeto, Unity no es un programa que disponga de muchas herramientas, por lo que no es muy apropiado para este cometido, y es necesario acudir a un programa secundario más adecuado para esto.

Existen varias plataformas de modelado 3D, de entre las cuales se ha optado por Cinema 4D, ya que es uno de los más populares existe una gran diversidad de bibliografía en internet orientada a aprender a usar este software. Otra de las razones de elección del mismo es la compatibilidad de Cinema 4D con Unity 3D, ya que permite exportar los objetos creados a Unity sin gran dificultad.

A continuación, se van a presentar los desarrollos de cada uno de los elementos de este trabajo. En los siguientes apartados se detallarán los modelos físicos y su funcionamiento y programación. En otra sección diferente se detallará el modelado 3d visual de los objetos físicos.

5.1. Cinta transportadora

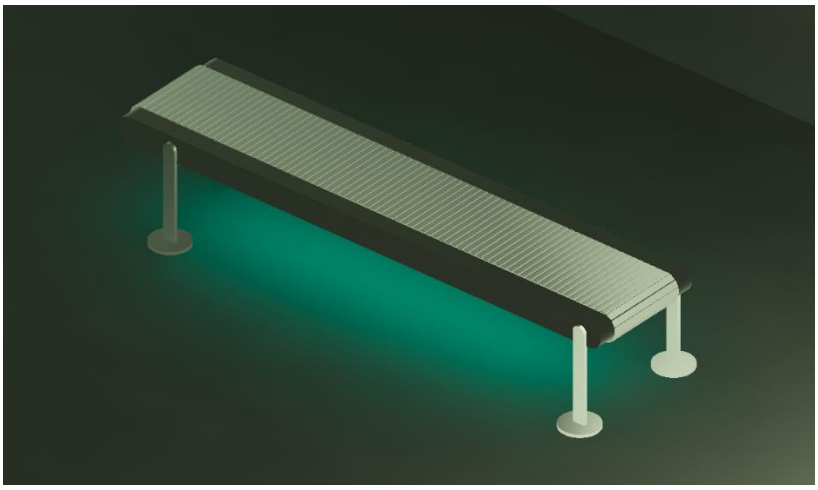

Figura 16. Visión general de la cinta transportadora

A la hora de modelar una planta industrial, la cinta transportadora constituye el elemento más común y necesario para el transporte de objetos.

Aunque en realidad para llevar a cabo es tarea se necesitan diversos elementos como motores, engranajes, rodillos, etc, desde el punto de vista de la simulación será tratada como placas (cuerpos físicos cúbicos), a las que se les añadirá una textura para asemejar la apariencia de una cinta transportadora común.

Sin embargo, es importante recalcar, tanto para este elemento como para el resto que se expondrán en este documento, que la interacción con los elementos dinámicos de la planta, es decir, las cajas que serán transportadas, será producida por dichas placas, a las que se les asignará, mediante código, el comportamiento de cada objeto, y no por la textura, que tiene un objetivo meramente gráfico y visual.

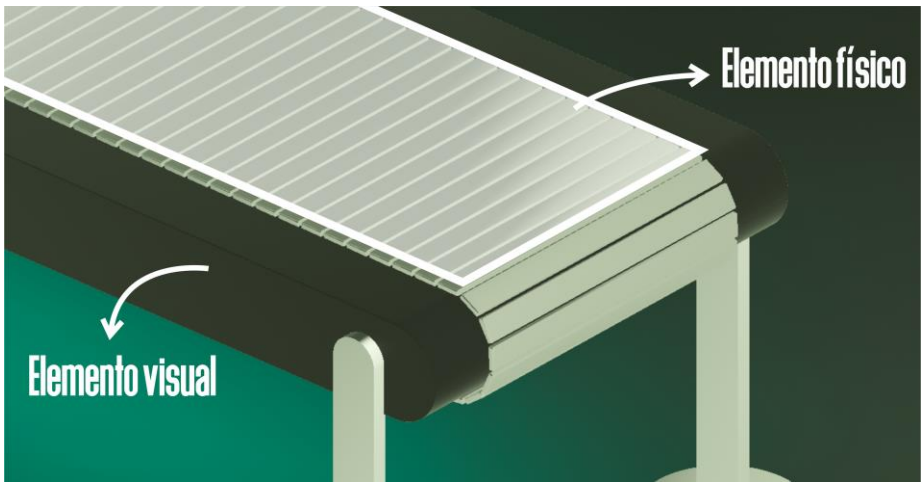

Figura 17. Elementos físico y visual de la cinta

El modelado de la cinta es un simple rectángulo del tamaño de la misma. A dicho objeto, mediante programación, se le asigna el comportamiento básico de una cinta transportadora: todo aquel objeto que entre en contacto con la misma adquirirá cierta velocidad en la dirección de la dimensión trasversal de la misma. Para conseguir esto, la solución se basa en asignar un pequeño desplazamiento trasversal a la cinta, y una vez realizado, asignar como nueva posición la inicial.

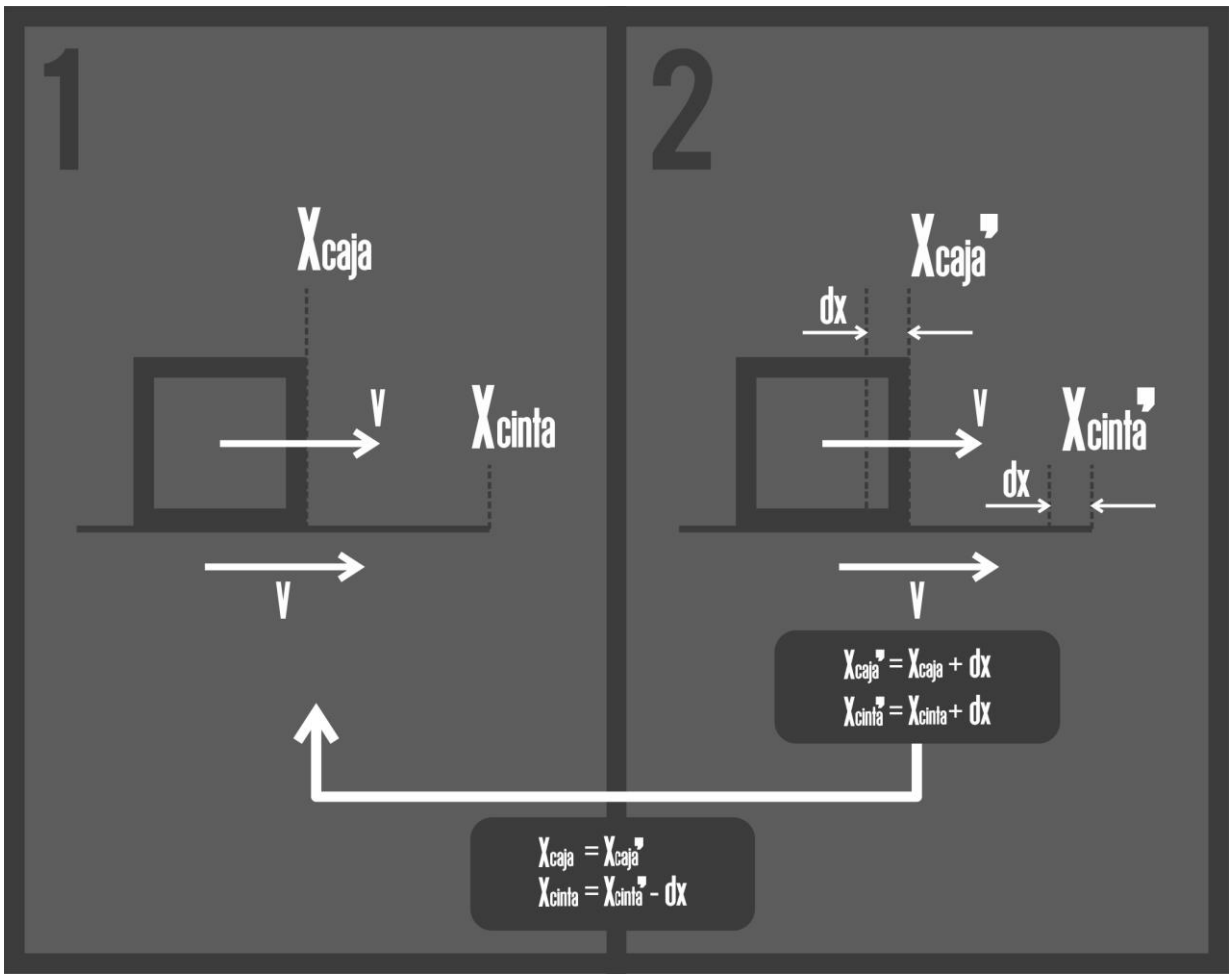

Figura 18. Esquema de movimiento de la cinta

Así, durante el movimiento inicial, todo lo que esté sobre la cinta se desplazará con ella, y cuando llegue al límite de desplazamiento, al asignar la posición inicial a la cinta de forma instantánea, dicha traslación no se trasmitirá a los objetos que estén encima. De esta forma, repitiendo esta rutina de forma cíclica, definiendo un desplazamiento pequeño para la cinta y una velocidad adecuada, el efecto que se aprecia es semejante al de una cinta transportadora, sin necesidad de modificar el comportamiento natural de los objetos dinámicos que se encuentren en la cinta, sino influir en su movimiento únicamente mediante la fricción con la cinta, asemejándose así lo más posible a la realidad.

```
cinta3.SetBool("flag", true);
Rigidbody = GetComponent<Rigidbody>();
rigidbody.position -= transform.forward * speed * Time.deltaTime;
rigidbody.MovePosition(rigidbody.position + transform.forward * speed * Time.deltaTime);
```

Con la función *transform.position* se asigna de forma instantánea la posición inicial de la plataforma, por lo que este no es propiamente un desplazamiento, y no afecta a lo que haya encima. Sin embargo, *rigidbody.MovePosition* es una función dirigida a un *rigidbody*, es decir, a un objeto con masa que está sujeto de alguna forma al motor físico. Esta función asigna un movimiento en una dirección determinada (en este caso en el eje x del objeto, que coincide con la dimensión trasversal de la plataforma) con una velocidad determinada. La velocidad, contenida en la variable speed, es multiplicada por el factor *Time.deltaTime*, que contiene el tiempo transcurrido entre dos fotogramas, dando como resultado la distancia que avanza el objeto por cada fotograma. Esta última función, al ser un desplazamiento de un objeto físico con una velocidad determinada, sí que afecta a los objetos apoyados sobre éste.

El resultado de la combinación de estas dos rutinas es una plataforma aparentemente en reposo que hace desplazarse a los objetos que se encuentran sobre ella, algo similar al comportamiento de una cinta transportadora.

5.2. Mesa giratoria

La mesa giratoria es un elemento típicamente diseñado para hacer una clasificación entre dos tipos de elementos, enviando cada uno de ellos hacia un lado u otro. Para el proceso de selección se hace uso de un sensor capaz de distinguir entre los dos tipos de objetos. Por ejemplo, se puede hacer una separación entre cajas grandes y pequeñas, colocando un sensor de presencia a una altura mayor que las cajas pequeñas, para solo detectar las grandes. Otra separación muy común es entre materiales metálicos y no metálicos, colocando un sensor electromagnético. El sentido de giro de la mesa giratoria estará determinado por la lectura del sensor. Otra función de la mesa giratoria, y es concretamente en la que se ha basado el autor de este proyecto, es girar la posición de las cajas al llegar a un giro, para mantener cada caja siempre con la misma orientación respecto a la cinta. Esto puede ser útil, por ejemplo, en el caso de que exista una etiqueta en la caja, colocada en el lateral, donde se vayan marcando, mediante software, los puestos de trabajo por los que va pasando, y así, colocando sensores, puede llevarse un cierto control de la trayectoria de dicha caja, y desecharla si no ha pasado por cierto puesto de trabajo o ha sido marcada como defectuosa. Este es un ejemplo real sacado de una planta automatizada de Renault.

En cuanto al modelado de la mesa, se trata de una placa cuadrada a la que se le han asignado dos rutinas:

- La misma rutina asignada a la cinta transportadora, para controlar el movimiento de la que contiene la mesa.
- La rutina de giro de la mesa, que imprime un giro de 90 grados a una velocidad concreta que puede ser modificada. En esta rutina se han añadido unos movimientos de subida y bajada de la plataforma, antes y después del giro, para evitar la colisión de la misma con las cintas adyacentes, debido a la proximidad entre ellas. La velocidad de subida y bajada de la mesa también puede ser manipulada por el usuario, eligiendo una apropiada a las especificaciones requeridas, así como el tipo de objeto transportado, no siendo conveniente una elevada velocidad si dicho objeto presenta cierta fragilidad.

Figura 19. Rutina de giro de la mesa

A continuación, se presenta el código referente a esta rutina, pero para hacer la explicación más entendible, se va a separar en dos bloques, el primero referente al giro de la mesa, y el segundo sobre la subida y bajada de la misma.

```
if(flag_girar && !flag_mesagirada2)
{
     Quaternion deltaRotation = Quaternion.Euler(rotation_axis * angularspeed * Time.deltaTime);
     mesa.MoveRotation(mesa.rotation * deltaRotation);
}

if (!flag_girar && !flag_mesagirada1)
{
     Quaternion deltaRotation = Quaternion.Euler(rotation_axis * (-angularspeed) * Time.deltaTime);
     mesa.MoveRotation(mesa.rotation * (deltaRotation));
}
```

En cuanto al giro de la mesa, se utilizan dos variables que indican el estado actual del giro, que son *flag_mesagirada1* y *flag_mesagirada2.* Cada una de estas variables se activan cuando la mesa se encuentra en cada una de las posiciones de giro, que son con un giro de 0º o con un giro de 90º.

A parte, se puede observar una tercera variable llamada *flag_girar.* Esta se utiliza para activar el giro de la mesa cuando se requiera. La lógica del código interpreta que si esta variable se activa (*flag_girar = TRUE*) corresponde a una petición de giro en un sentido. Si la variable se desactiva (*flag_girar = FALSE*) indica una petición de giro en sentido contrario.

Por tanto, para que la mesa gire de la posición 1 a la 2 deben cumplirse que *flag_mesagirada2* esté desactivada, y que *flag_girar* tenga el valor *TRUE.*

Por el contrario, para que gire en el sentido opuesto, *flag_mesagirada1* debe estar desactivada, y *flag_girar* con valor *FALSE.*

Una vez que se cumplen las condiciones de giro, la función capaz de crear la rotación es *mesa.MoveRotation,* que se encarga de rotar en cada fotograma del juego un ángulo determinado. Dicho ángulo se calcula es el calculado en *deltaRotation*.

```
if (flag_subir && !flag_mesasubida)
                                                                                                {
  mesa.MovePosition(mesa.position + transform.up * speed * Time.deltaTime);
}

if (!flag_subir && !flag_mesabajada)
                                                                                                {
  mesa.MovePosition(mesa.position - transform.up * speed * Time.deltaTime);
                                                                                                }
```

Para la subida y bajada se utiliza la misma estructura que para el giro. La variables que indican el estado de la mesa aquí son *flag_mesasubida* y *flag_mesabajada*. La variable que indica la petición de subida/bajada es *flag_subir*, de tal forma que la condición de subida de la mesa es que *flag_subir sea TRUE* y que

flag_mesasubida esté desactivada, y para la bajada, *flag_subir* tiene que ser *FALSE,* y *flag_mesabajada* debe estar desactivada.

Si las condiciones se cumplen, la función *mesa.MovePosition* hace que la mesa se desplace verticalmente, con velocidad contenida en la variable *speed.*

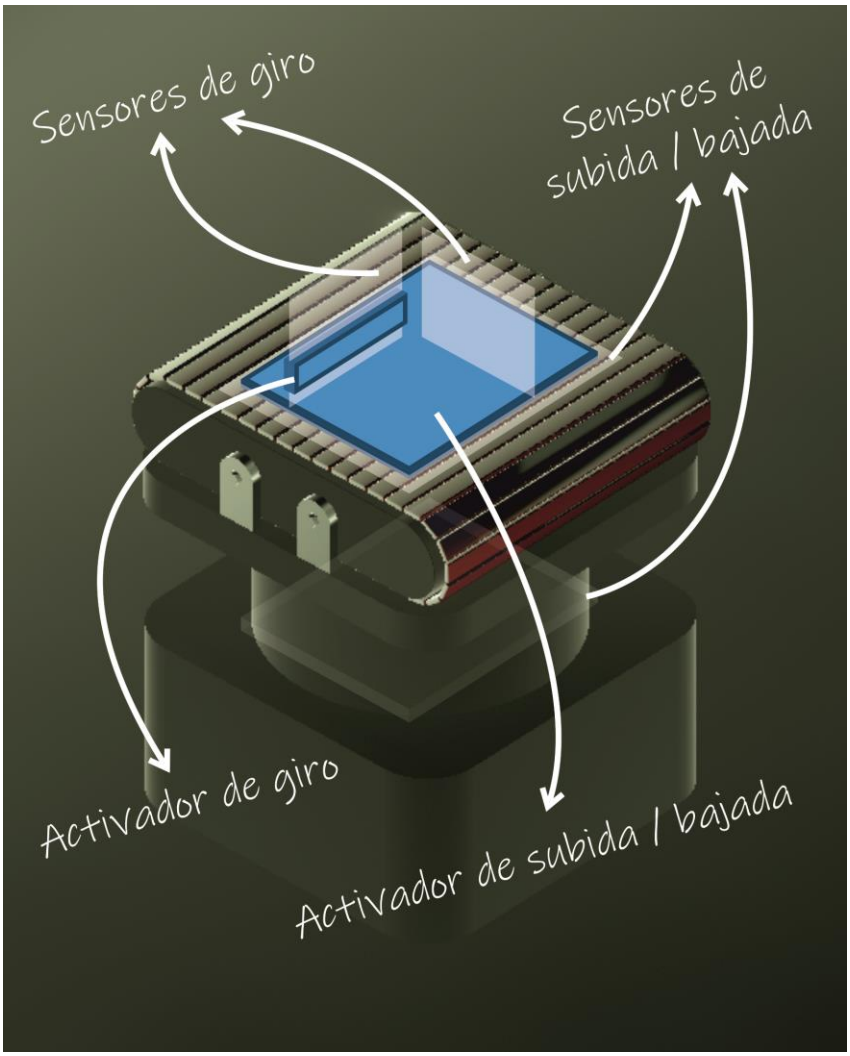

Figura 20. Elementos físicos y sensores de la mesa giratoria

Para detectar el fin de la subida, bajada y giro en ambos sentidos, se ha recurrido al uso de objetos configurados de tal forma que, al contactar con otros objetos concretos, activen una variable asociada. Este tipo de configuración de un objeto es lo que se conoce como disparador o ***trigger.***

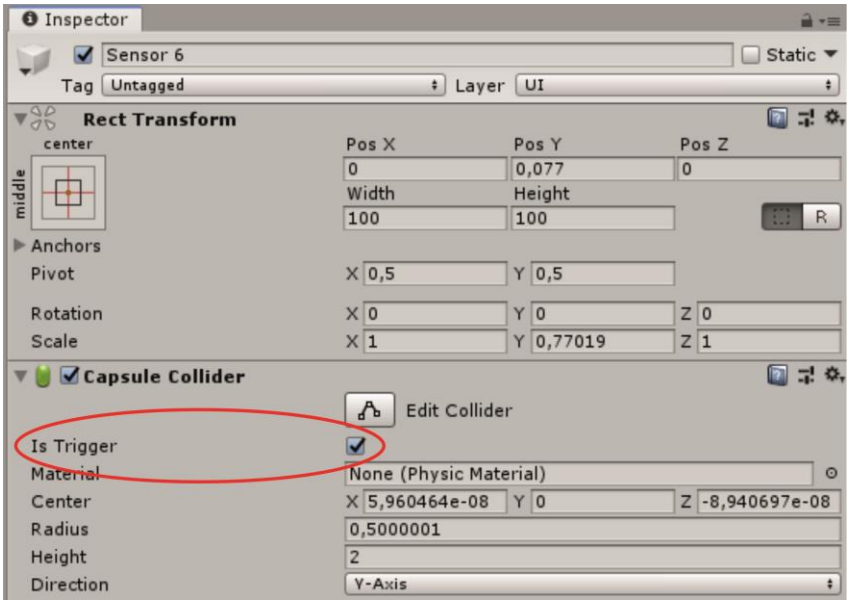

Figura 21. Activación de la configuración de trigger en un objeto

Esta configuración se activa dentro de la ventana del inspector. Al activar esta función, tiene un efecto doble en el objeto:

- Deshabilita el *collider* del objeto, es decir, desaparecen las colisiones con otros objetos. Al configurarse como *trigger*, dicho objeto no opondrá resistencia al entrar en contacto con otro objeto, y lo atravesará.
- Al contactar con otros objetos, se activará una variable de *trigger*, que puede ser leída, haciendo que el objeto pase a comportarse como un sensor de presencia. Esta es la principal función del *trigger*. De esta forma, se puede simular el comportamiento de sensores, y usarlos para rutinas que requieran cierto *feedback* sobre el fin de un desplazamiento, la presencia de un objeto en una posición concreta, etc. En el caso de la mesa, estos *triggers* van conectados a las variables que indican los estados de la mesa, que son *flag_mesasubida, flag_mesabajada, flag_mesagirada1* y *flag_mesagirada2.*

5.3. Grúa pórtico

Este elemento realiza la misma función que la cinta transportadora, es decir, trasladar de forma lineal un objeto. Sin embargo, lo hace de una forma muy distinta, mediante fuerza de agarre en vez del transporte por fricción. Este cambio da acceso a nuevas posibilidades que pueden ser muy útiles y ventajosas para determinados casos.

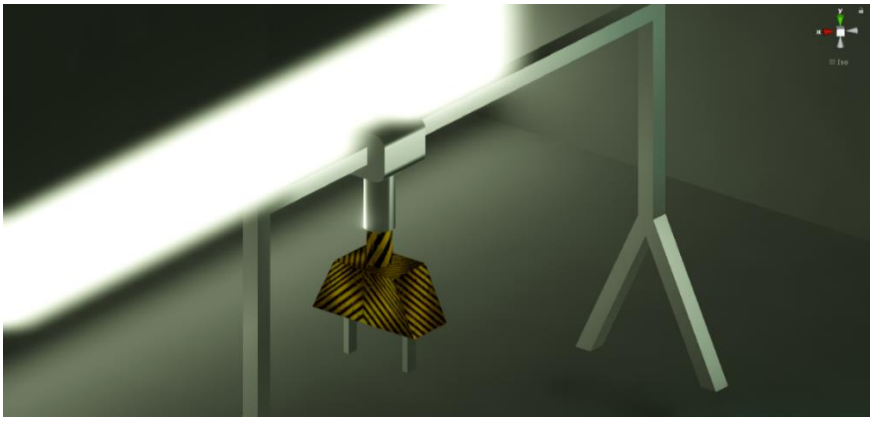

Figura 22. Visión general de la grúa pórtico

Poniendo un ejemplo, este modelo de transporte hace posible que el objeto transportado pueda ser colocado sobre otra pieza, al final del desplazamiento, ya que la grúa eleva el objeto antes de desplazarlo. Por tanto, determinando la posición final de la grúa sobre la pieza secundaria, se logran dos acciones con un mismo elemento: el desplazamiento y la unión con la otra pieza. Otro fin por el que sería útil emplear este modo de desplazamiento pueden ser ciertas restricciones de espacio, o de estructuración de la planta, que no permite la colocación de una cinta en dicho lugar. Sin embargo, este elemento cuenta con numerosas desventajas respecto a la cinta, como el elevado costo, o el modo de agarre, que debe ser adecuado al objeto que se va a transportar. Esta configuración cuenta con numerosas restricciones, y por tanto suele utilizarse únicamente en casos en los que sea estrictamente necesario.

El comportamiento de este elemento se divide en dos, que son el movimiento trasversal y vertical de la grúa, y la apertura y cierre de las garras.

Para el desarrollo de esta rutina de comportamiento es importante tener en cuenta que existen 4 posiciones claves de la grúa, que son las combinaciones resultantes a la grúa desplazada a la izquierda o la derecha, barra extendida (abajo) o recogida (arriba). En casos como éste, donde existen pocas posiciones clave y bien definidas, es mucho más sencillo acudir a una herramienta llamada “animator controller”, en la que se definen varios estados, y en cada uno de ellos se vincula una posición de la grúa, o la transición de una a otra. Cada una de estas posiciones y animaciones se han creado en Cinema4D, el mismo software donde se han modelado los elementos, que también cuenta con la posibilidad de crear animaciones. Este tipo de animaciones son útiles y adecuadas de utilizar cuando el movimiento de dicha animación no dependa de ningún elemento externo, ya que se trata de una animación que no puede programarse de tal forma que sea sensible a interacciones externas, como por ejemplo detener un movimiento cuando se detecte una colisión con otro elemento. En este caso será necesario acudir a la programación de dicha animación. Tanto el movimiento trasversal como el vertical de la grúa no dependen de ningún elemento externo, por lo que podemos acudir a esta solución. Sin embargo, para la apertura y cierre de las pinzas es necesaria la detección de la caja, por lo que esta rutina ha de ser programada desde cero.

5.3.1. Desplazamiento trasversal

El desplazamiento lateral se compone de dos posiciones claves, que son la grúa a la derecha y a la izquierda. Se trata de un movimiento sencillo que, al no depender de ningún objeto dinámico externo, ha sido realizando haciendo uso del *animator controller*.

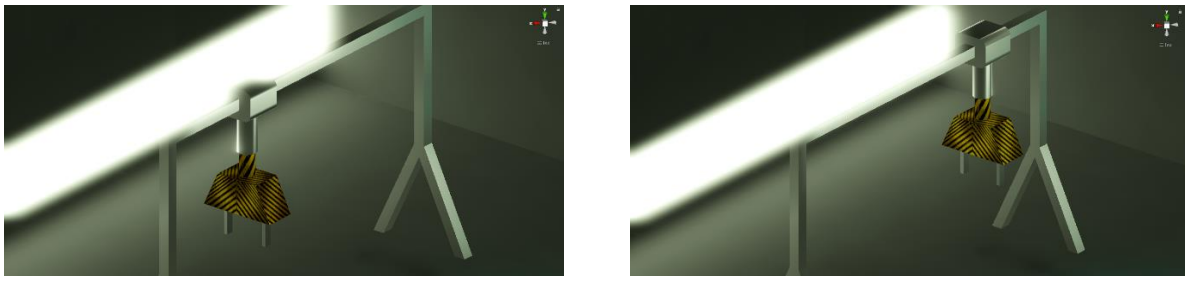

Figura 23. Desplazamiento horizontal de la grúa

El diagrama de flujo de esta rutina son cuatro estados unidos por transiciones con condiciones de paso de un estado a otro. Los estados 1 y 3 contienen las posiciones clave, denominadas *Pos 1* y *Pos 2*. Los estados 2 y 4 contienen las animaciones de la grúa desplazándose de una posición a otra. Estas animaciones son creadas en Cinema4D y exportadas a Unity 3D.

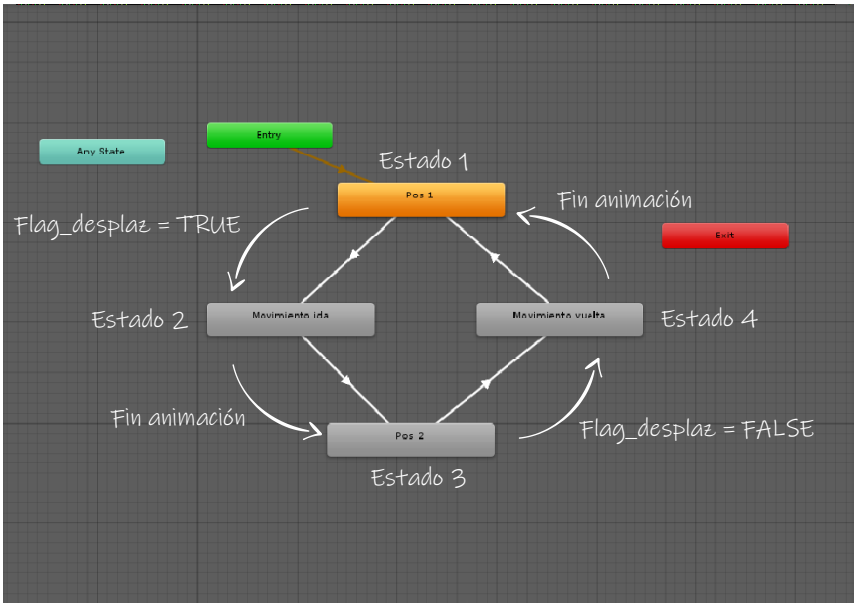

Figura 24. Diagrama de estados de la rutina de movimiento horizontal de la grúa programado en el animator controller

El paso de los estados 2 y 4 a los estados 1 y 3 tiene la condición de esperar al fin de la animación. Por tanto, el sistema estará en estos estados únicamente durante la duración de la animación. Una vez que alcanza la nueva posición, el sistema pasa al estado 1 o 3, y permanece en ese estado hasta que se active la condición, que en este caso es la activación de la variable *Flag_desplaz.*

5.3.2. Desplazamiento vertical

Este desplazamiento de ha diseñado de la misma forma que el anterior. En este caso las posiciones clave son la grúa contraída y extendida, es decir, arriba y abajo.

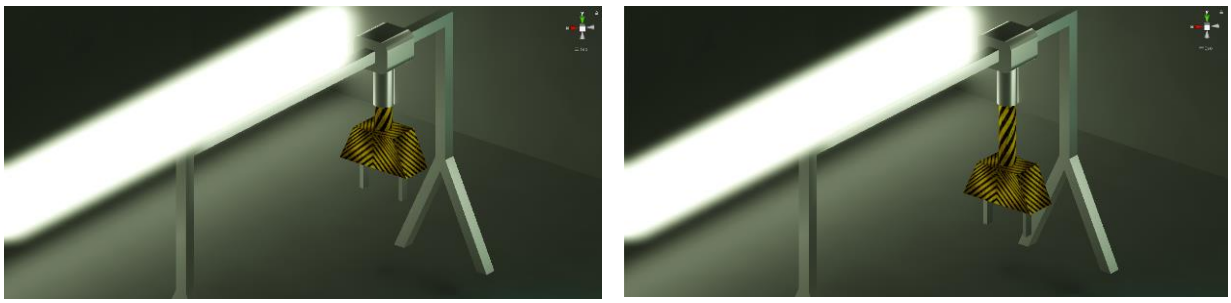

Figura 25. Desplazamiento vertical de la grúa

La extensión de la grúa se ha diseñado para que llegue a agarrar la caja que venga por la cinta.

A continuación, se muestra el diagrama de estados del *animator controller* correspondiente a esta rutina.

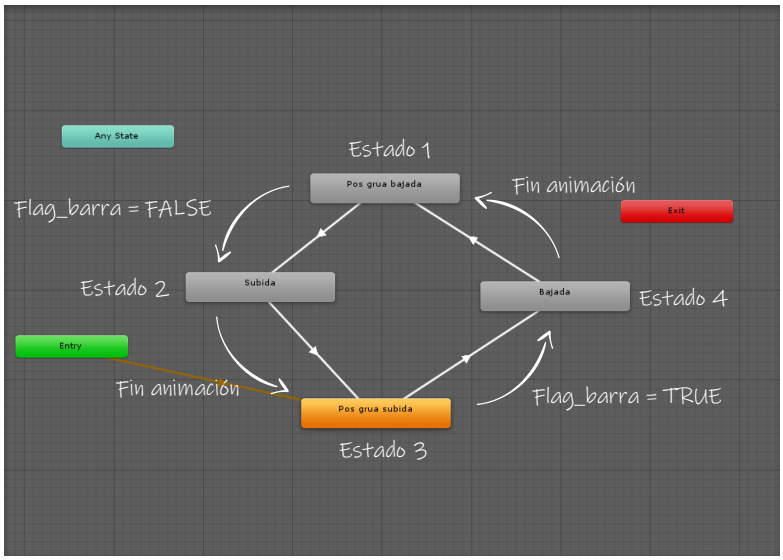

Figura 26. Diagrama de estados de la rutina de movimiento horizontal de la grúa programado en el animator controller

En este caso, la condición de paso de los estados 1 y 3 a los estados 2 y 4, es decir, la condición de cambio de posición, depende de la variable *Flag_barra*. La velocidad de subida y bajada puede definirse en la ventana del inspector referente al *animator controller*, modificando el tiempo de duración de la animación.

Figura 27. Modificación de la velocidad de la animación

5.3.3. Apertura y cierre de pinzas

Este proceso es el encargado de agarrar la caja que llega por la cinta y evitar que caiga cuando la grúa se eleve, y soltarla cuando se desplace hasta la posición final.

Se trata de un proceso delicado, ya que el objetivo es que la grúa ofrezca la posibilidad de agarrar cajas de tamaños variados, y detener el cierre de las pinzas cuando detecte que ha contactado con la caja. Este proceso necesita la programación de ciertos *sensores* capaces de detectar la caja. Esto supone que el uso del *animator controller* no es adecuado en este caso, ya que es un movimiento dependiente de la interacción con la caja.

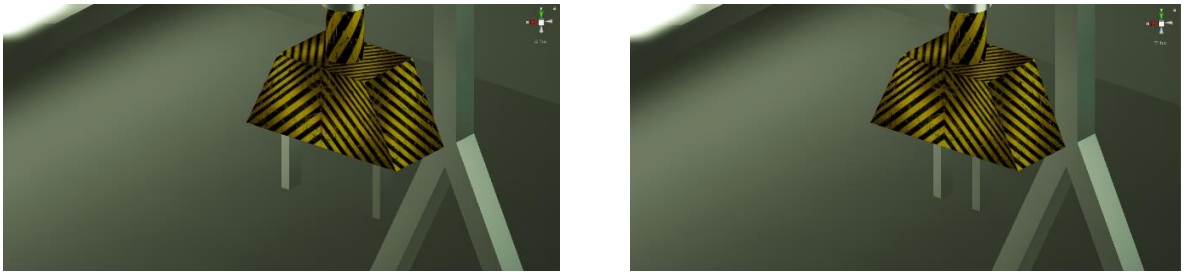

Figura 28. Apertura y cierre de pinzas de la grúa

Además, para conocer el estado de las pinzas (abiertas o cerradas, con caja y sin caja) también se han creado objetos a modo de sensores.

5.3.4. Sensores detectores de caja

Para detectar la caja al cerrar las pinzas, se han creado dos *placas* que se han configurado como *triggers* y se han ubicado en la parte interior de cada pinza, ya que esa va a ser la superficie de contacto con la caja.

Es importante matizar que la posición de las placas respecto a las pinzas debe ser constante, y no modificarse cuando se muevan dichas pinzas. Para ello, se han situado jerárquicamente dentro de cada una de las pinzas, es decir, se han hecho *hijas* de cada pinza. Esto se realiza en la ventana *Hierarchy,* arrastrando el gameobject *hijo* sobre el que se desea que sea su *padre.* Esta nueva relación hace que los parámetros locales del *hijo* respecto del *padre* sean constantes, como la posición y orientación.

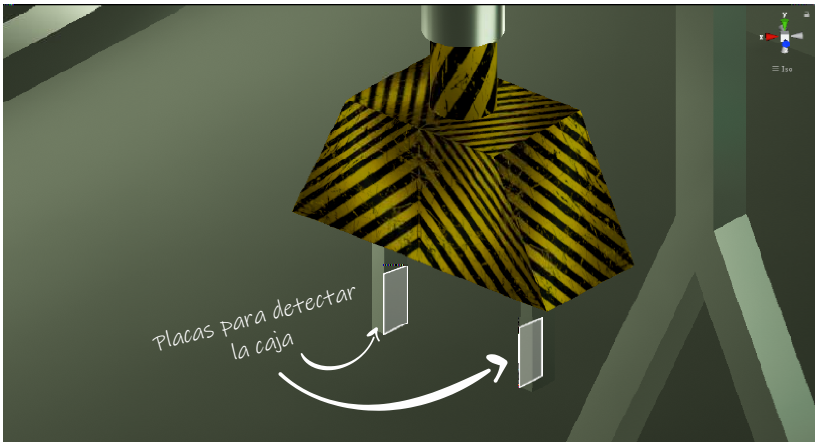

Figura 29. Elementos físicos de la grúa para detectar la caja

El script asociado a cada placa se llama *Fin_empuje*, y será un script utilizado también por los pistones de encuadre de cajas, como se explicará más adelante.

```
public class Fin_empuje : MonoBehaviour
{
    public bool caja = false;
    public GameObject caja_actual;
    public Rigidbody caja_actual_fisica;
    public bool fin_pinza1 = false;
    public bool fin_pinza2 = false;

    // Start is called before the first frame update
    void OnTriggerStay(Collider col)
    {

        if (col.gameObject.name.Contains( "Caja"))
        {
            caja = true;
            caja_actual = col.gameObject;
            caja_actual_fisica = col.attachedRigidbody;
```

```
            }

        }

        void OnTriggerExit(Collider col)
        {
            if (col.gameObject.name.Contains("Caja"))
            {
                caja = false;
                caja_actual = null;
                caja_actual_fisica = null;
            }
        }

    }
```

Aquí se emplean dos funciones muy comunes a la hora de trabajar con triggers, que son *OntriggerStay* y *OnTriggerExit*. Estas son funciones que se ejecutan cuando hay interacción entre el trigger y otro objeto.

La primera, *OnTriggerStay,* se ejecuta **mientras** haya un objeto atravesando la superficie del trigger. Por tanto, siempre que haya una caja tocando la superficie del trigger, la variable *caja* de dicha pinza tomará el valor *TRUE.*

La segunda, *OnTriggerExit*, se ejecuta en el momento en el que un objeto que ha entrado en la zona del trigger deja de estar en contacto. Es decir, como su nombre indica, cuando el objeto *sale* del trigger. En este caso, cada vez que esto ocurra, la variable *caja* de la pinza correspondiente se pondrá a *FALSE.*

Además, en este script puede apreciarse que, al activarse el trigger, si el *collider* correspondiente (el objeto con el que ha interaccionado) se trata de una caja, dicha caja es asignada a la variable *caja_actual*, y su componente *rigidbody* con la variable *caja_actual_fisica*. Esto es necesario porque, para el momento de levantar dicha caja, se tendrán que modificar algunos parámetros de dicha caja para hacer posible la traslación.

5.3.5. Sensores de apertura y cierre de pinzas

Por otro lado, se han creado varios sensores para detectar la apertura y cierre total de las pinzas, para conocer el estado de las mismas. Esta rutina se realizó previamente usando un contador que iba almacenando la distancia recorrida, y se comparaba con dos parámetros correspondientes a las pinzas abiertas y cerradas. Sin embargo, este método no era muy fiable, y fue sustituido por la creación de sensores, así se evita el uso de decimales, y la programación es más sencilla.

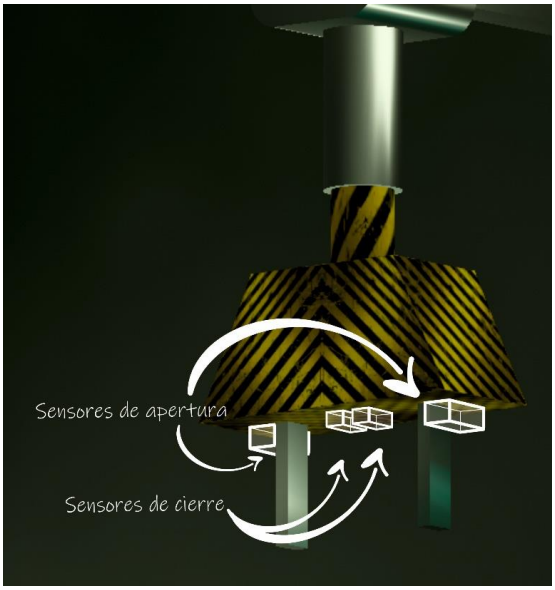

Figura 30. Sensores de apertura y cierre de pinzas de la grúa

Como puede observarse en la imagen, existen 4 sensores, dos de apertura y dos de cierre. Los 4 contienen el mismo *script* llamado *Sensor*. Este *script* se ha procurado que sea lo más genérico posible para poder ser usado por todos los sensores, a excepción de aquellos que necesiten realizar alguna operación concreta, como por ejemplo *Fin_empuje*, ya explicado previamente.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sensor : MonoBehaviour
{
    public bool flag;
    // Start is called before the first frame update
    private void OnTriggerStay(Collider other)
    {
        flag = true;
    }
    private void OnTriggerExit(Collider other)
    {
        flag = false;
    }
}
```

Este *script* es bastante sencillo. Lo único que hace es crear una variable llamada *flag*, que se activa y desactiva cuando lo haga el trigger del *gameobject* asociado.

Posteriormente, la variable *flag* es leída por el objeto que necesita de dicho sensor, en este caso las dos pinzas, y asignan el valor de la variable a la que corresponda en su *script*, concretamente *pinza1_abierta, pinza1_cerrada, pinza2_abierta* y *pinza2cerrada.*

5.3.6. Programa asociado a las pinzas

Una vez explicado el funcionamiento de los sensores de cada pinza, se describe el código asociado a cada una de las pinzas que, de forma resumida, será aquel que gestione la información recibida por los sensores, y actuará en consecuencia. Además, hay ciertas acciones o información que depende de ambas pinzas, como el momento en el que ambas han contactado con la caja. Esta gestión se desarrolla en la pinza 1, y la información resultante es enviada a la pinza 2. Por tanto, se explica primero el *script* de la pinza 1, y luego se hará algún apunte de la pinza 2.

En primer lugar, se declaran todas las variables necesarias.

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public class Pinza1 : MonoBehaviour
    {

        // Declaración variables ------------------------------------------------------------------------

        public Control controlvar;
        public float speed = 0.1f;
```

```
public bool flag;
public bool parada;
public Pinza2 pinza2var;
public Transform guia;
public GameObject caja_actual;
public Rigidbody caja_actual_fisica;
public Fin_empuje empujepinza1var;
public bool caja1;
public bool caja2;
public Sensor aperturapinza1var;
public Sensor cierrepinza1var;
public bool pinzas_abiertas, pinzas_cerradas, pinza1_abierta, pinza2_abierta, pinza1_cerrada, pinza2_cerrada;
Rigidbody pinza1;

// ------------------------------------------------------------------------------------
```

La primera variable declarada, *controlar,* recoge todas las variables generadas en el programa *Control*, rutina que coordina todos los elementos para automatizar el proceso completo, que se explicará más adelante. *Control* utiliza una variable para dar la señal de activación a las pinzas. Esa variable es *pinzas_grua,* que en la rutina de la pinza 1 se almacena en la variable *flag,* como puede apreciarse en la parte de asignación de variables, mostrada a continuación.

```
void Update()
{

// Asignación variables ------------------------------------------------------------------

    controlvar = GameObject.Find("Controlobject").GetComponent<Control>();
    flag = controlvar.pinzas_grua;

    empujepinza1var = GameObject.Find("Detectorpinza1").GetComponent<Fin_empuje>();
    caja1 = empujepinza1var.caja;
    caja_actual = empujepinza1var.caja_actual;
    caja_actual_fisica = empujepinza1var.caja_actual_fisica;

    aperturapinza1var = GameObject.Find("Tope_apertura_pinza1").GetComponent<Sensor>();
    pinza1_abierta = aperturapinza1var.flag;

    cierrepinza1var = GameObject.Find("Tope_cierre_pinza1").GetComponent<Sensor>();
    pinza1_cerrada = cierrepinza1var.flag;

    pinza2var = GameObject.Find("pinza 2").GetComponent<Pinza2>();
    caja2 = pinza2var.caja2;
    pinza2_abierta = pinza2var.pinza2_abierta;
    pinza2_cerrada = pinza2var.pinza2_cerrada;

    pinza1 = GameObject.Find("pinza 1").GetComponent<Rigidbody>();

// ------------------------------------------------------------------------------------
```

En esta parte, se almacenan todas las variables generadas por los sensores en variables propias de la *Pinza 1.*

Hay alguna variable que necesita información de ambas pinzas, como son *pinzas_abiertas, pinzas_cerradas* y *parada*. Su valor se asigna a continuación.

```
// Condiciones generación variables globales ----------------------------------------------------

    // Condición pinzas abiertas

    if (pinza1_abierta && pinza2_abierta)
    {
        pinzas_abiertas = true;
    }
    else
    {
        pinzas_abiertas = false;
    }

    // Condición pinzas cerradas

    if ((pinza1_cerrada && pinza2_cerrada) || parada == true)
    {
        pinzas_cerradas = true;
    }
    else
    {
        pinzas_cerradas = false;
    }

    // Condiciones parada

    if (caja1 && caja2)
    {
        parada = true;
    }

    if (parada && !flag)
    {
        parada = false;
    }

//-------------------------------------------------------------------------------------------
```

Una vez terminada la declaración y asignación de variables, es el momento de definir las condiciones y la acción de apertura y cierre de la pinza 1, que será similar a la pinza 2.

En primer lugar, las condiciones de apertura de la pinza 1 son :

1. Que llegue la señal de apertura de la rutina *Control* (*flag == FALSE*).
2. Que no esté abierta (*pinza1_abierta == FALSE*)

Si estas condiciones se cumplen, la pinza comienza a desplazarse en la dirección del eje x de sus coordenadas locales, es decir, en la dirección de apertura.

```
// Apertura y cierre de pinza 1 ----------------------------------------------------------------

    // Apertura pinza 1

    if (flag == false && pinza1_abierta == false)
    {
         pinza1.MovePosition(pinza1.position + transform.right * speed * Time.deltaTime);
    }
```

De forma análoga, las condiciones de cierre son:

1. Que llegue la señal de cierre de la rutina *Control* (*flag == TRUE*).
2. Que no esté cerrada (*pinza1_cierre == FALSE*).
3. Que no haya interacción con una caja (*parada == FALSE*).

Si estas condiciones se cumplen, la pinza se desplaza en la dirección del eje x y sentido negativo de sus coordenadas locales, es decir, en la dirección de cierre.

```
    // Cierre pinza 1

    if (flag == true && parada == false && pinza1_cerrada == false)
    {
         pinza1.MovePosition(pinza1.position - transform.right * speed * Time.deltaTime);
    }
```

En el caso de encontrar una caja en la trayectoria de cierre, cuando ambas pinzas la detectan, se activa la variable de parada. Entonces su movimiento cesa, y se ejecutan ciertas instrucciones para adaptar la caja para su agarre, que son la desactivación de la gravedad para dicha caja, y la configuración de la misma como *hija* de las pinzas. Realmente es *hija* de el elemento *guía,* que es un objeto vacío situado en el centro de las pinzas, que no tiene escala. Así se evita que la caja modifique su escala, modificando solo la posición.

```
    // Adaptación caja para levantamiento

    if (parada)
    {
        guia.position = caja_actual.transform.position;
        caja_actual.transform.SetParent(guia.transform);
        caja_actual.transform.position = guia.transform.position;
        caja_actual_fisica.useGravity = false;
    }
```

Figura 31. La caja deja de ser hija de la grúa en el momento en que las pinzas se abren al soltarla en la posición final

Una vez trasladada la caja y abiertas las pinzas, se restaura la fuerza de la gravedad en la caja, y se elimina su relación con las pinzas, para continuar su recorrido.

```
            // Adaptación caja después de levantamiento


            if (!parada)
            {
                caja_actual.transform.parent = null;
                caja_actual_fisica.useGravity = true;


            }
        }

        //-----------------------------------------------------------------------------------------
    }
```

Estas medidas fueron necesarias para simular el agarre de la caja por parte de las pinzas, aunque no reflejan un comportamiento realista, ya que se elimina la fuerza del peso.

Esta relación de objetos es muy útil ya que, cuando se programe el movimiento de la pinza, no habrá que preocuparse también del movimiento del sensor, ya que permanecerá con las mismas coordenadas locales respecto a la pinza.

En cuanto al código de la pinza 2, es exactamente igual, salvo que las variables *pinzas_abiertas, pinzas_cerradas* y *parada* son directamente leídas de la rutina de pinza 1, aunque de todas estas, sólo se usará la variable *parada.* El resto son usadas por la rutina *control.*

Tampoco existe en esta rutina las instrucciones asociadas a la adaptación de la caja para el momento del agarre y transporte, ya que también se encarga de esto la rutina de la pinza 1.

A continuación, se muestra, por tanto, el código asociado a la pinza 2.

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class Pinza2 : MonoBehaviour
{
    // Declaración variables ---------------------------------------------------------------------

    public float speed = 0.5f;
    public bool flag;
    public bool caja;
    public bool parada;
    public Pinza1 pinza1var;
    public Fin_empuje empujepinza2var;
    public bool caja2;
    public Sensor aperturapinza2var;
    public bool pinza2_abierta;
    public Sensor cierrepinza2var;
    public bool pinza2_cerrada;
    Rigidbody pinza2;

    // ------------------------------------------------------------------------------------------

    void Update()
    {

    // Asignación variables ----------------------------------------------------------------------

        aperturapinza2var = GameObject.Find("Tope_apertura_pinza2").GetComponent<Sensor>();
        pinza2_abierta = aperturapinza2var.flag;

        cierrepinza2var = GameObject.Find("Tope_cierre_pinza2").GetComponent<Sensor>();
        pinza2_cerrada = cierrepinza2var.flag;

        empujepinza2var = GameObject.Find("Detectorpinza2").GetComponent<Fin_empuje>();
        caja2 = empujepinza2var.caja;

        pinza1var = GameObject.Find("pinza 1").GetComponent<Pinza1>();
        parada = pinza1var.parada;
        speed = pinza1var.speed;

        flag = pinza1var.flag;

        pinza2 = GameObject.Find("pinza 2").GetComponent<Rigidbody>();

    //------------------------------------------------------------------------------------------

    // Apertura y cierre de pinza 2 ---------------------------------------------------------------

        // Apertura pinza 2

        if (flag == false)
        {
```

```
                if (pinza2_abierta == false)
                {
                    pinza2.MovePosition(pinza2.position + transform.right * speed * Time.deltaTime);
                }
            }

            // Cierre pinza 2

            if (flag == true && parada == false)
            {
                if (pinza2_cerrada == false)
                {
                    pinza2.MovePosition(pinza2.position - transform.right * speed * Time.deltaTime);
                }
            }
        }

        //-------------------------------------------------------------------------------------------
    }
```

5.4. Sensores láser

Después de haber explicado la funcionalidad del trigger y el uso de objetos como sensores, este elemento va a ser bastante sencillo de entender.

Un sensor láser se sirve de la emisión y recepción de un haz de luz láser para detectar la presencia de objetos dentro de su espacio de cobertura.[7]

El haz de luz es generado por el elemento emisor de luz (láser) en el transmisor, y es detectado por el elemento de recepción de luz (receptor).

Por su configuración se pueden diferenciar varios tipos:

Modelo reflectivo: Tanto el emisor de luz como los elementos receptores están contenidos en una sola carcasa. El sensor recibe la luz reflejada desde el objeto.

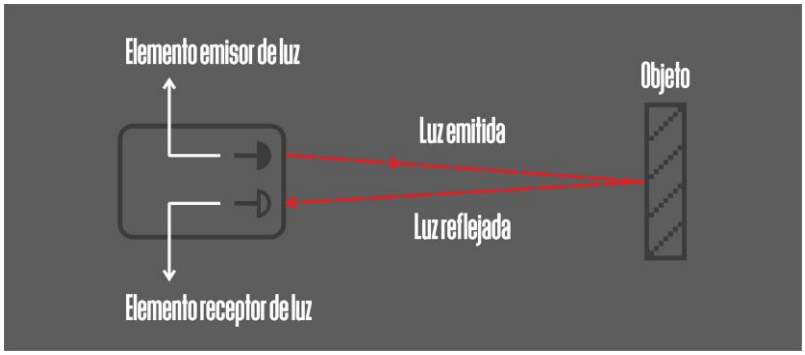

Figura 32. Esquema del modelo reflectivo del sensor láser

Modelo de barrera: El transmisor y el receptor están separados. Cuando el objeto se encuentra entre el transmisor y el receptor, se interrumpe la luz.

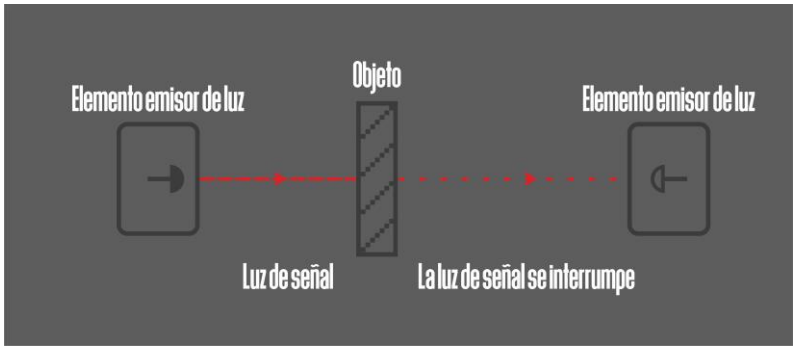

Figura 33. Esquema del modelo de barrera del sensor láser

Modelo retro reflectivo: Tanto el emisor de luz como los elementos receptores están contenidas en un mismo recinto. La luz del elemento emisor incide en el reflector y regresa al elemento receptor de luz. Cuando hay un objeto presente, se interrumpe la luz.

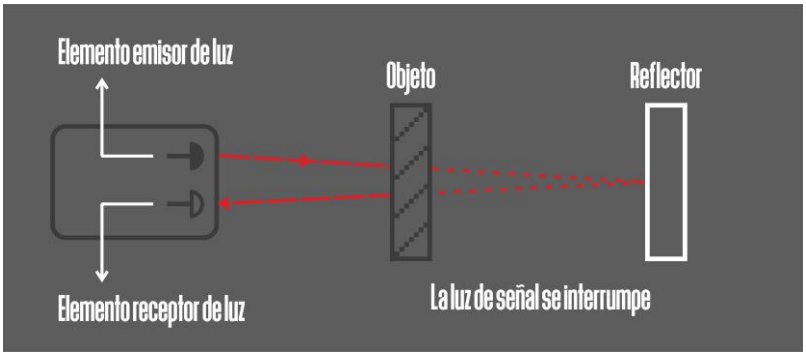

Figura 34. Esquema del modelo retroreflectivo del sensor láser

El sensor que se ha diseñado para este proyecto es el de tipo barrera, ya que es el que más se ajusta a las necesidades de la planta.

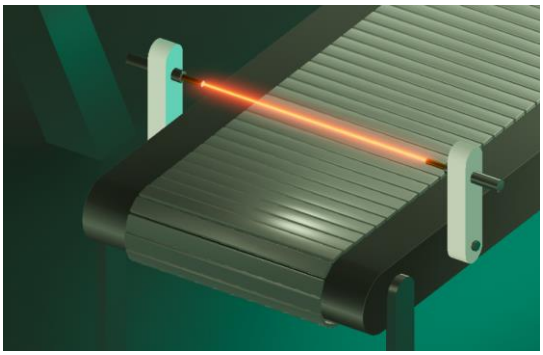

Figura 35. Visión general del sensor láser del proyecto

Se han creado varios sensores colocados al inicio y al final de los distintos elementos. Estos sensores son muy útiles para poder automatizar toda la planta, ya que, mediante la lectura de su variable asociada, puede saberse en qué posición se encuentra la caja, y así activar o desactivar el elemento que proceda.

La parte del sensor que interactúa con el resto de objetos es el rayo láser. Los dos soportes laterales, que harían la función de emisor y receptor, han sido diseñados en Cinema 4D, pero son meramente decorativos, es decir, no tienen ninguna programación asociada. Para modelar el rayo láser, se ha creado un cilindro de radio muy pequeño y longitud del ancho de la cinta, y se ha configurado como *trigger*, para hacer las funciones de sensor. La apariencia de luz se ha implementado añadiéndole un nuevo material de color rojo y configurado como luminiscente.

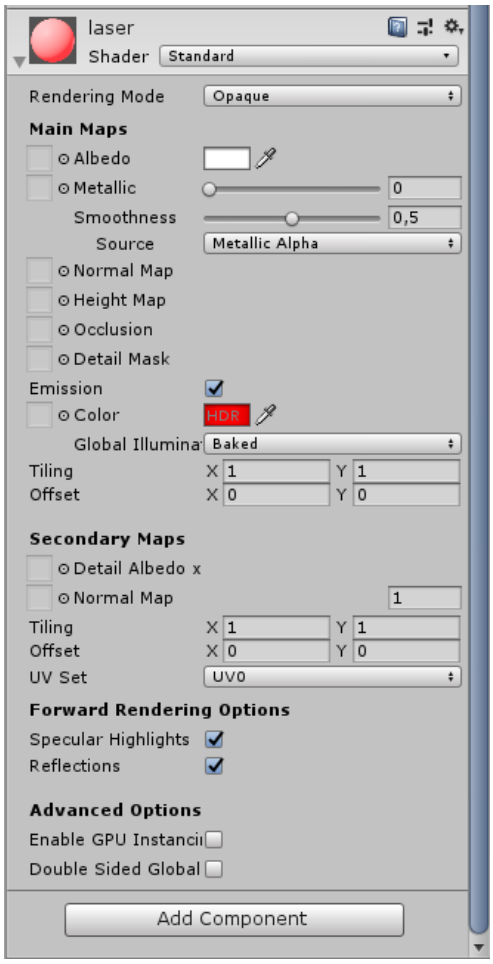

Figura 36. Configuración de material luminescente

El código asociado al sensor se llama *SensorIU*. Este *script* es exactamente el mismo que *sensor,* ya explicado anteriormente, con la diferencia de que se le ha añadido la funcionalidad de mostrar un indicador por pantalla que se enciende y apaga conforme a la variable asociada.

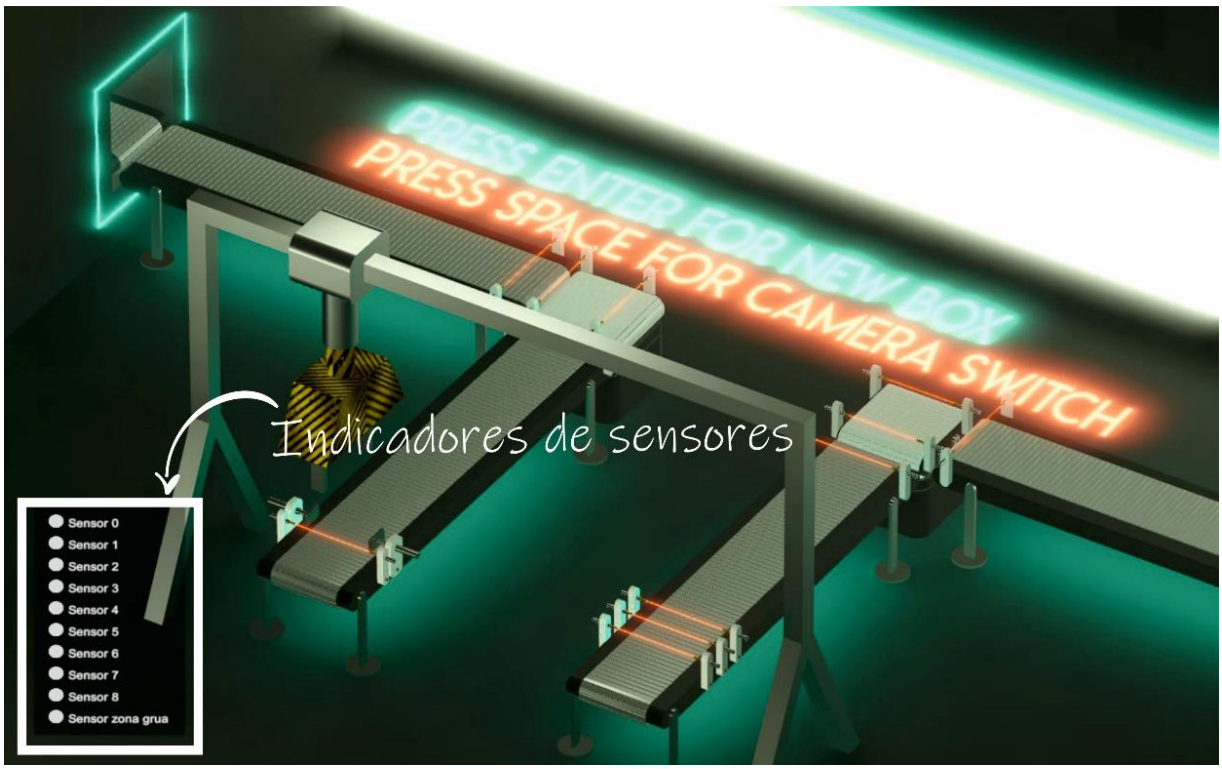

Figura 37. Panel de información de estado de los sensores

Para crear estos indicadores, seleccionar la opción *Create > UI.* En este apartado se encuentran todas las opciones de creación de la *Interfaz del Usuario (User Interface).* Esta es la sección destinada a la creación de menús de nuestra aplicación, así como botones e indicadores adicionales, como en nuestro caso.

En primer lugar, se ha creado un panel donde se van a ubicar todos los indicadores. Éste se encuentra en *Create > UI > Panel.* En cuanto se crea el panel, se creará otro *gameobject* de forma adicional, del que dependerá el panel, llamado *Canvas.* Este objeto es el que define la superficie que abarca la interfaz del usuario, donde deben situarse todos los objetos que pertenezcan a ésta. Por tanto, todos ellos deben ser *hijos* del objeto *Canvas.*

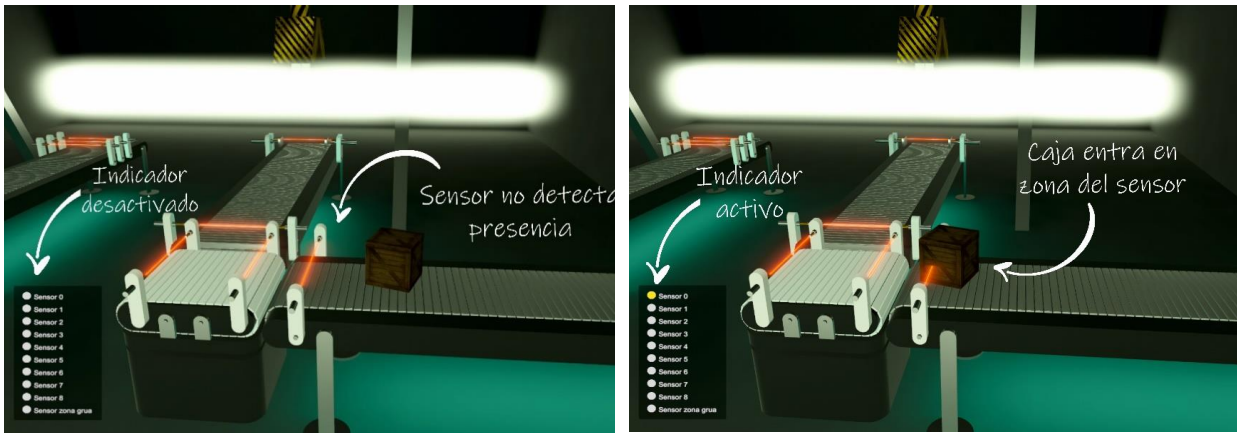

Figura 38. Activación del sensor 0 por parte de una caja

Una vez creado el panel, en el inspector aparecen todos los parámetros que definen su apariencia, donde se le ha modificado el color y la trasparencia, para que no tape el fondo, y se vea a través de dicho panel.

A continuación, es necesario crear los indicadores. Éstos no son más que imágenes que cambian dependiendo del valor de la variable asociada. Para ello, seleccionar *Create > UI > Image.* Al crear este *gameobject,* será necesario, en el inspector, asociar la imagen a mostrar. Para poder seleccionar dicha imagen dentro de Unity, se ha almacenado dentro de una nueva carpeta ubicada dentro del fichero de datos del proyecto, llamada *Resources*. El siguiente paso es buscar dicha carpeta en nuestro proyecto, y al abrirla se mostrarán las imágenes allí guardadas. Para poder utilizar las imágenes en el *UI,* se ha cambiado el formato a *Sprite (2D and UI).* Una vez hecho esto, es cuestión de arrastrar dicha imagen al recuadro *Source image.*

Sin embargo, para reflejar el estado de los sensores, no es necesaria una, sino dos imágenes.

Las que se han usado son un círculo grisáceo y otro amarillo, simulando un led encendido y apagado, respectivamente.

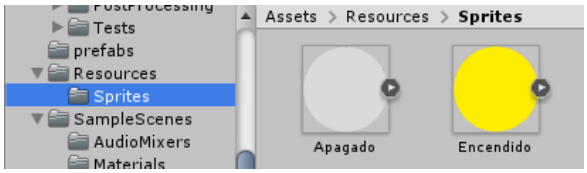

Figura 39. Indicadores de sensor encendido y apagado creados y almacenados en la carpeta Resources

Así pues, será necesario cierta programación para asignar una imagen u otra, dependiendo del valor de la variable asociada al sensor. Todo esto se recoge en el script *sensor_UI,* explicado a continuación.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SensorUI : MonoBehaviour
{
    // Start is called before the first frame update
    public bool flag = false;
    public Image UIimagen;
    public GameObject indicador;

    public void Start()
    {
        UIimagen = indicador.GetComponent<Image>();
    }
```

En primer lugar, después de añadir todas las librerías necesarias, entre las que se encuentra *UnityEngine.UI,* se crean las variables necesarias, y se asigna a *UIimagen* el valor del parámetro *imagen* del indicador.

A continuación se describe el mismo código que el recogido en *sensor,* es decir, la activación y desactivación de la variable dependiendo de la interacción con el *trigger* del sensor.

```
    void OnTriggerStay(Collider other)
    {
        flag = true;
    }

    void OnTriggerExit(Collider other)
    {
        flag = false;
    }
```

Por último, se describe la asignación de cada imagen en base a la misma variable. Esta rutina debe ejecutarse en cada fotograma, ya que la pantalla se recalcula una vez por fotograma.

```
    public void Update()
    {
        if (flag == true)
        {
            UIimagen.sprite = Resources.Load<Sprite>("Sprites/Encendido");
        }

        else
        {
            UIimagen.sprite = Resources.Load<Sprite>("Sprites/Apagado");
        }
    }
}
```

Así, cuando el sensor indique la presencia de una caja, su indicador asociado se iluminará, y se mantendrá encendido hasta que la caja atraviese su zona de acción.

5.5. Pistones de encuadre de cajas

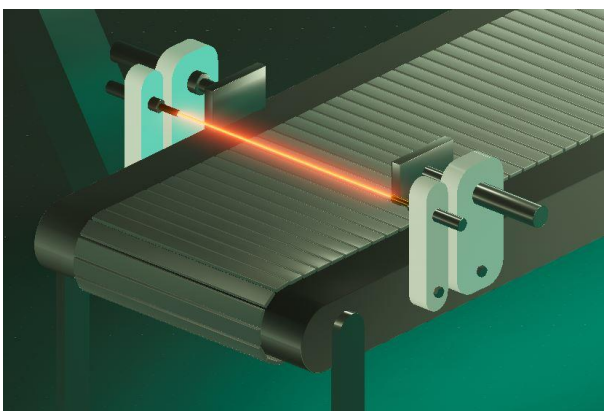

Figura 40. Visión general de los pistones de encuadre

Estos elementos se han añadido a este prototipo de planta con el objetivo de cuadrar la caja que entra en posición para ser atrapada por la grúa, para corregir la orientación de la misma, y evitar problemas de agarre.

Ambos pistones comienzan a acercarse, empujando la caja, y no se detendrán hasta que la caja se oriente de forma paralela a los mismos, es decir, formando un ángulo de 90 grados.

Para ello, se pensó primeramente ubicar un sensor en cada una de las superficies de contacto de cada pistón, de forma similar a las pinzas de la grúa.

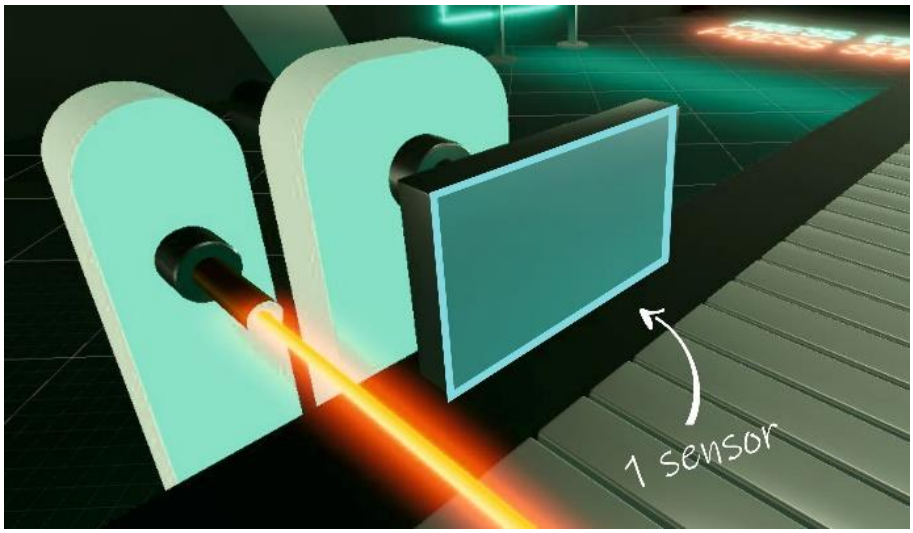

Figura 41. Un sensor de detección de cajas del pistón, ocupando toda la superficie del mismo

Sin embargo, durante el planteamiento de la resolución de este comportamiento se detectó el hecho de que dos sensores no son suficientes para lograr el encuadre de la caja, ya que en el momento de contactar ambos pistones con la caja se detendría el empuje del mismo, no alcanzando necesariamente la orientación deseada.

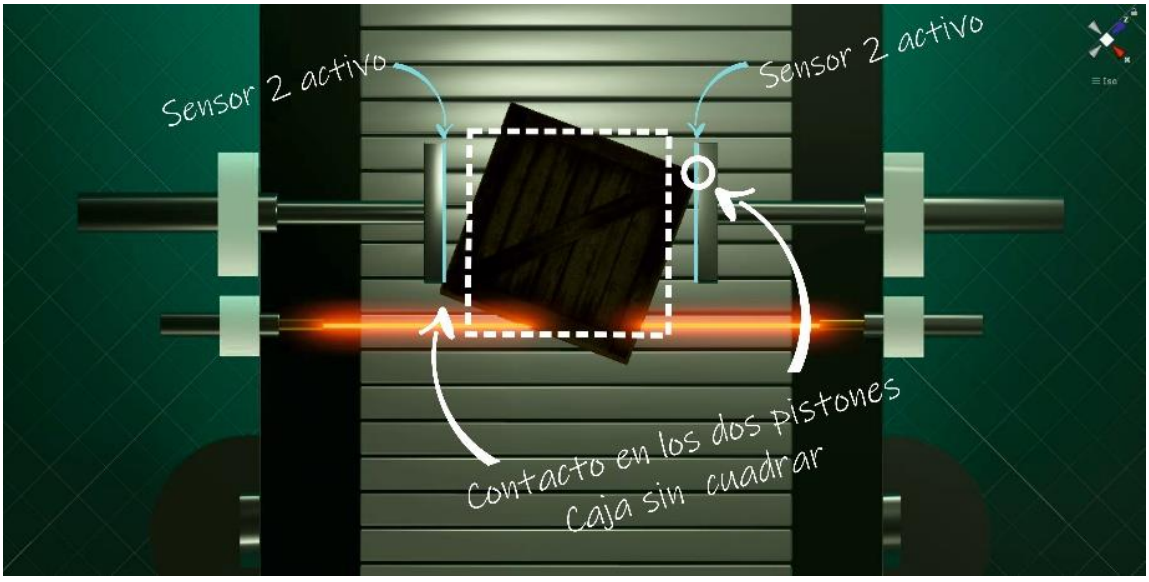

Figura 42. Los pistones se detienen antes de cuadrar completamente toda la caja debido a la existencia de un único sensor por pistón

Para resolver este detalle, se ha procedido a dividir cada sensor en dos, de manera que ahora se cuenta con 4 sensores, dos por cada pistón, situando cada uno en un extremo de la superficie del pistón.

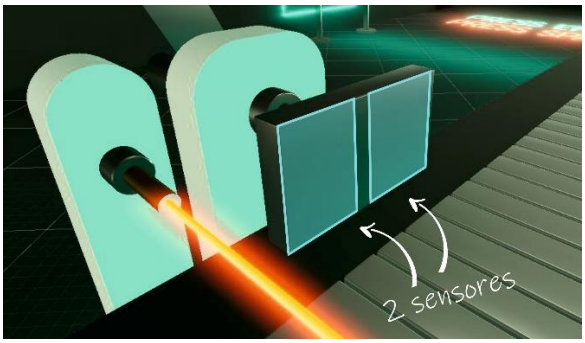

Figura 43. Dos sensores de detección de cajas del pistón, ocupando cada uno la mitad de la superficie del pistón

La condición de paro del empuje de los pistones ahora es la detección de una caja por parte de los 4 sensores de forma simultánea. Este simple cambio ahora obliga a los pistones a seguir empujando hasta que la caja adopte una orientación tal que contacte con los cuatro sensores. Esto sólo se conseguirá si está totalmente enderezada, formando un ángulo recto con la dirección de empuje de los pistones.

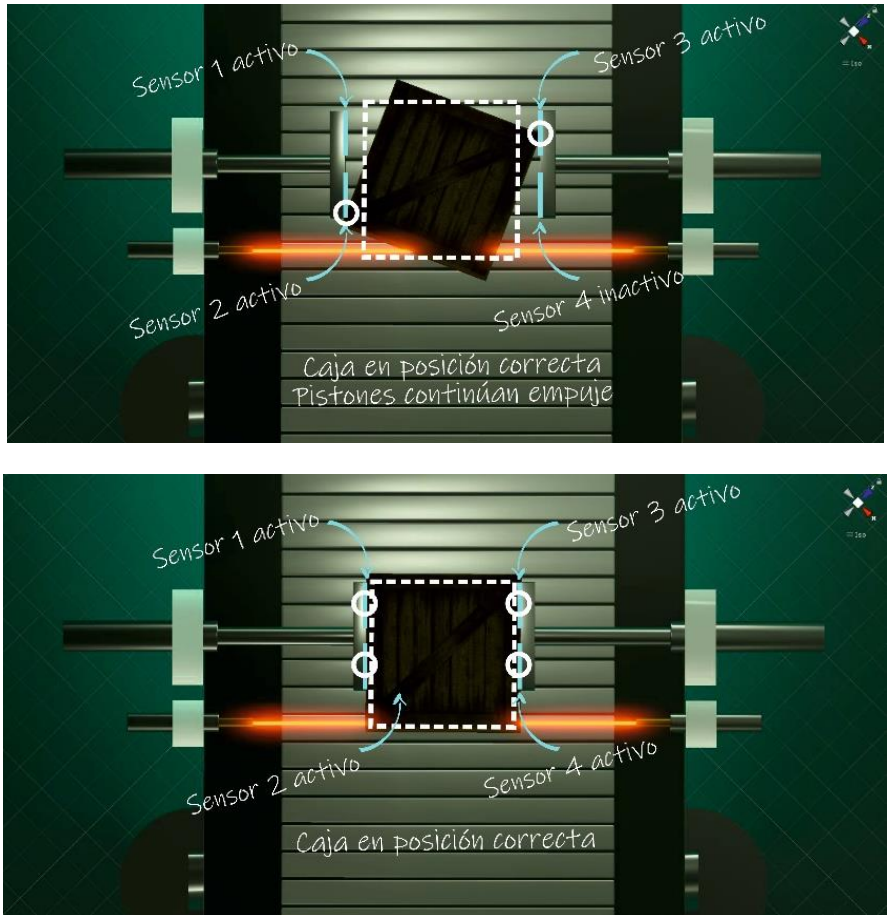

Figura 44. La caja queda completamente cuadrada al ubicar dos sensores en cada pistón

5.6. Creador de cajas

Este elemento tiene la función de gestionar la llegada de nuevas cajas por el túnel de entrada, conforme a la decisión del usuario, cada vez que presiona la tecla *Intro.*

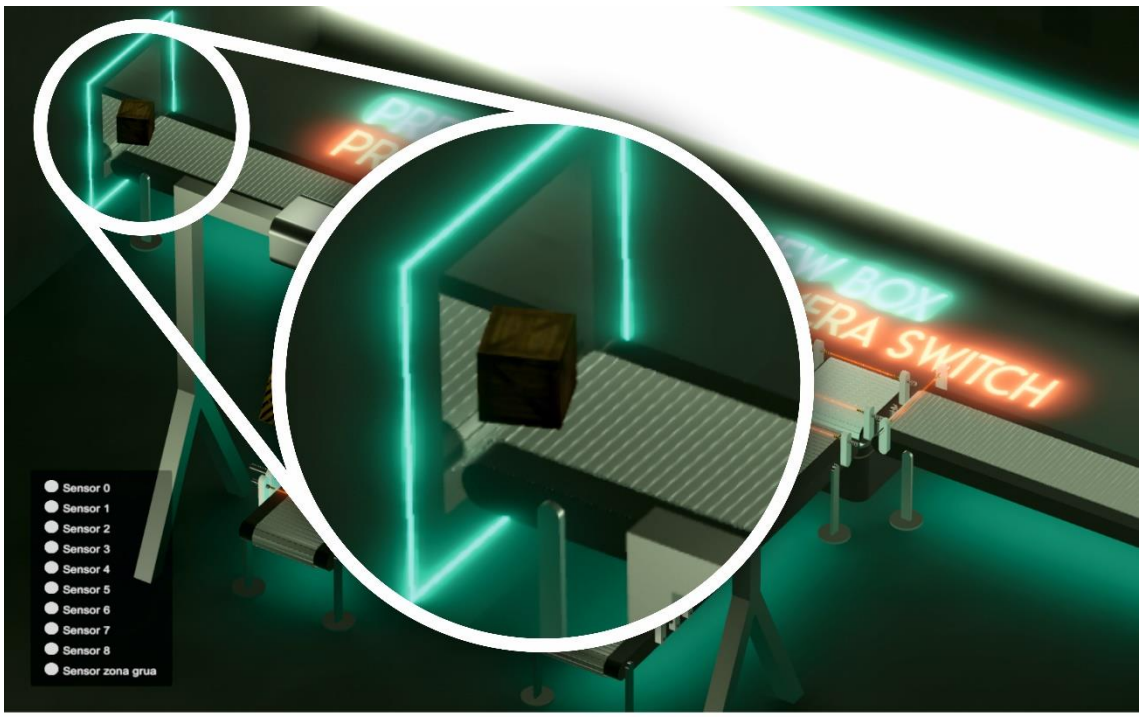

Figura 45. Aparición de nueva caja por el túnel de entrada de la planta

Para crear este túnel, se ha creado un hueco al que se ha añadido un segundo cubo del tamaño deseado, que hace las funciones de túnel. En el interior del mismo se ha colocado la cinta transportadora de entrada, y un *gameobject* llamado *Spawn_cajas*, cuyo script asociado, *Creadorcajas,* es el encargado de la generación de cajas.

Además, se ha situado un cubo transparente en la misma posición del objeto *Spawn_cajas*, configurado como sensor, que será el encargado de habilitar la creación de una nueva caja cuando no haya ninguna en el espacio reservado para la misma. Dicho sensor se denomina *Habilitadornuevacaja.*

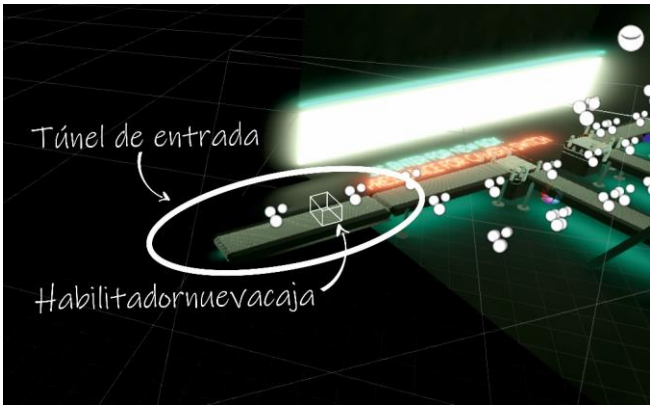

Figura 46. Visión del sensor Habilitadornuevacaja situado en el túnel de entrada

A continuación, se explica *Creadorcajas,* script asociado al *gameobject Spawn cajas,* código encargado de crear una nueva caja en la posición indicada cada vez que el usuario apriete la tecla *Intro*.

```
public class Creadorcajas : MonoBehaviour
{
    public Transform spawnpos;
    public Rigidbody nueva_caja;
    public Sensor Habilitadornuevacajavar;
    public bool flag;
    public bool flagboton;
    public Quaternion rotation;
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown("enter"))
            flagboton = true;
        if(flagboton)
        {
            Habilitadornuevacajavar = GameObject.Find("Habilitadornuevacaja").GetComponent<Sensor>();
            flag = Habilitadornuevacajavar.flag;

            if (!flag)
            {
                rotation=
Quaternion.Euler(spawnpos.transform.rotation.x,Random.Range(0f,360f),spawnpos.transform.rotation.z);
                Instantiate(nueva_caja, spawnpos.transform.position, rotation);
            }
            flagboton = false;
        }

    }
}
```

El programa es bastante sencillo. En primer lugar, la variable *Spawnpos* almacena la posición y orientación inicial que toma cada nueva caja. Sin embargo, la orientación respecto al eje Y se genera de forma aleatoria, usando la función *RandomRange(a,b),* donde *a* y *b indican los valores mínimo y máximo* que puede tomar el número aleatorio generado por dicha función. Como se trata de un ángulo, el rango será de 0 a 360 grados. Esta orientación aleatoria se ha pensado para poner a prueba los pistones de encuadre, y así observar su funcionalidad, aunque en la realidad lo normal será que las cajas lleguen todas con una orientación similar, y la corrección será mínima.

La variable *Habilitarnuevacajavar* es la que almacena la información procedente del sensor *Habilitarnuevacaja*, de la cual procede la variable *flag*, que indica si está libre la posición de salida para la creación de una nueva caja.

Además, la variable *flagboton* almacena la pulsación de la tecla *Intro*, mediante la función *Input.GetKeyDown("enter").* Cada vez que se activa esta variable, se comprueba que la posición de salida está libre, es decir, que *flag == FALSE*. Si esto se cumple, mediante la función *Instantiate* se crea una nueva caja, diseñada previamente y almacenada como *prefab*.

5.7. Destructor de cajas

Este es último elemento funcional que compone la planta, y aunque no es visible, es muy necesario para el correcto funcionamiento de toda la planta para evitar saturación.

La función del destructor, como su nombre indica, es la eliminación de las cajas que completan el recorrido de la planta.

Es un objeto necesario y propio únicamente del simulador, ya que en la realidad no tiene sentido. Su funcionamiento es bastante sencillo.

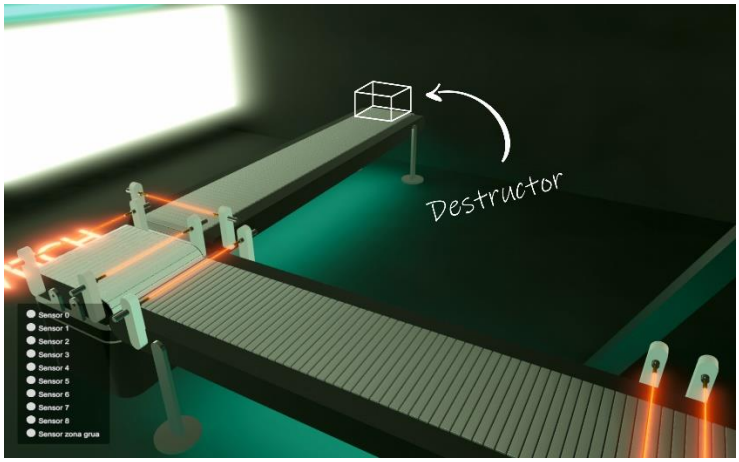

Figura 47. Elemento destructor de cajas situado al final del recorrido de la planta

Se trata de un cubo al que se le ha quitado la visibilidad, y configurado como trigger. Cuando alguna caja entra en contacto con dicho objeto, la caja es eliminada del programa.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Destructor : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        Destroy(other.gameObject);
    }
}
```

Para ello se usa la función *Destroy,* que se encarga de eliminar el objeto considerado como *Other,* definido en la función *OntriggerEnter* como aquel que entra en contacto con el *trigger.*

El objeto *Destructor* se coloca al final de la última cinta transportadora, para que las cajas que entren en contacto sean aquellas que han finalizado el recorrido de la planta.

6 DESCRIPCIÓN DETALLADA DE LOS ELEMENTOS VISUALES 3D

Una vez explicada la parte física, en este apartado se va a explicar el modelado 3D de cada uno de elementos anteriores. Esta es la parte visual que observa el usuario, pero no afecta al comportamiento del mismo, ni interacciona con los demás. Simplemente trata de asemejar la apariencia del objeto real.

Para ello, en este proyecto se ha hecho uso del programa denominado Cinema 4D.

Cinema 4D es un software para modelado 3D, renderizado y animación que ha ganado una considerable popularidad en los últimos años. Su uso tiene un amplio espacio dentro de la industria de la televisión y la publicidad gracias a sus características específicas para motion graphics.

Cinema 4D es un programa de modelado 3D, animación y renderizado que ha ganado importancia e interés en los últimos años en el ámbito del diseño y el *marketing* debido a su fácil aprendizaje y sus numerosas ventajas, destacando sus características específicas para *motion graphics*, es decir, para la generación de animaciones de objetos.

Este programa destaca tanto por sus posibilidades de generar contenido de gran calidad a la par de un fácil aprendizaje, gracias a una interfaz intuitiva. Su usabilidad es envidiable en relación con sus competidores y sin duda es el programa que ofrece la mejor interfaz gráfica.

Entre las muchas ventajas que se pueden enumerar de este software destaca el llamado MoGraph, un poderoso conjunto de herramientas que pueden hacer gráficos en movimiento de manera tremendamente sencilla. Sus características permiten, por ejemplo, clonar objetos, agregar efectos y crear movimiento de manera más fácil y rápida.

En definitiva, Cinema 4D es un software 3D fácil de aprender y muy potente. Tanto los principiantes como los profesionales experimentados pueden beneficiarse de la amplia gama de herramientas y funciones de Cinema 4D para alcanzar rápidamente resultados impresionantes en sus escenas 3D. [8]

Es por ello que se ha elegido dicho software para el diseño visual de los elementos de este proyecto. A continuación se explica de forma detallada la creación de cada uno de ellos.

6.1. Cinta transportadora

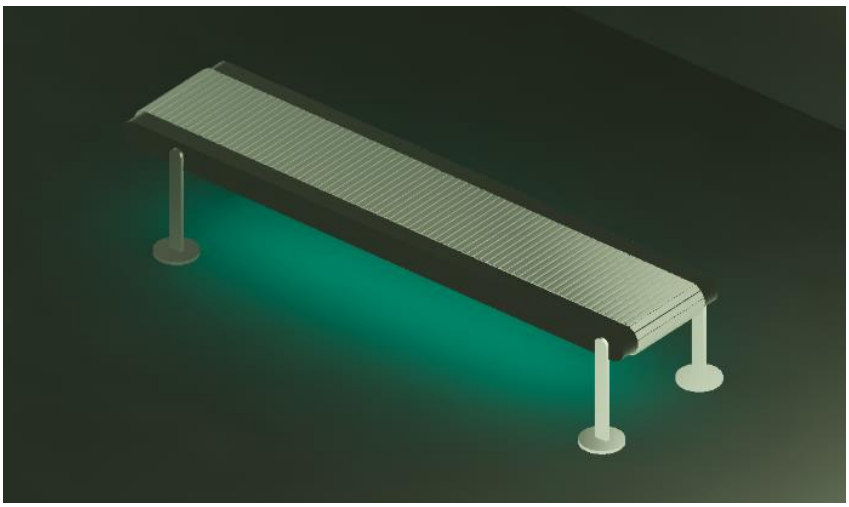

Figura 48. Apariencia final de la cinta transportadora

El diseño de la cinta transportadora se divide en varias partes, que al final se unen para formar un único objeto.

En este caso, se divide en el soporte central, las 4 patas y la cinta móvil. A continuación, se explica cada parte por separado, para facilitar su comprensión.

6.1.1. Soporte central

Esta primera pieza es bastante sencilla, ya que la única dificultad se encuentra en el diseño del perfil del mismo. Una vez definida la forma, basta con aplicar una extrusión en el eje perpendicular al perfil, con la longitud deseada para el ancho de la pieza.

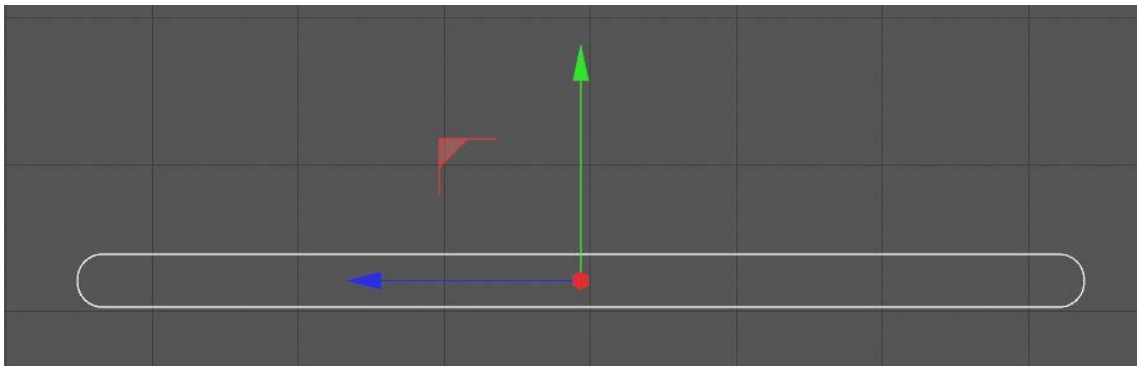

Figura 49. Silueta del perfil del soporte central de la cinta

Para conseguir la forma del perfil, se crea un rectángulo del tamaño deseado. En este caso, mide 689cm x 36cm.

La curva de los extremos se forma utilizando la opción de *redondear,* introduciendo el radio de giro deseado, en este caso 18 cm, es decir, la mitad del ancho del rectángulo.

Con estos pasos ya estaría definida la forma del perfil. Para darle volumen, se usa la herramienta

Extrusión. Al seleccionar esta herramienta, se creará en el explorador de archivos, y para aplicarla sobre el perfil, es necesario que dicho perfil esté ubicado dentro de dicha extrusión.

Figura 50. Uso de la herramienta Extrusión

Una vez aplicada, se configura la longitud de dicha extrusión, 155 cm.

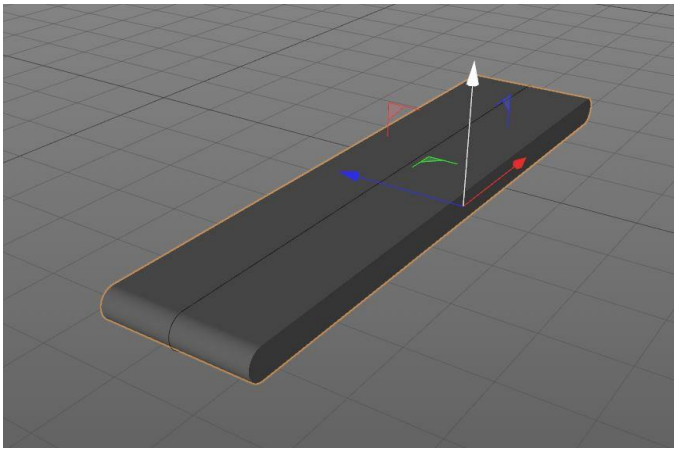

Figura 51. Soporte central de la cinta tras haberle aplicado volumen

Para terminar, se han añadido dos tapas en los laterales de la extrusión para crear un borde redondeado, con el fin de crear unos bordes mejor definidos al aplicar cierta iluminación sobre el objeto.

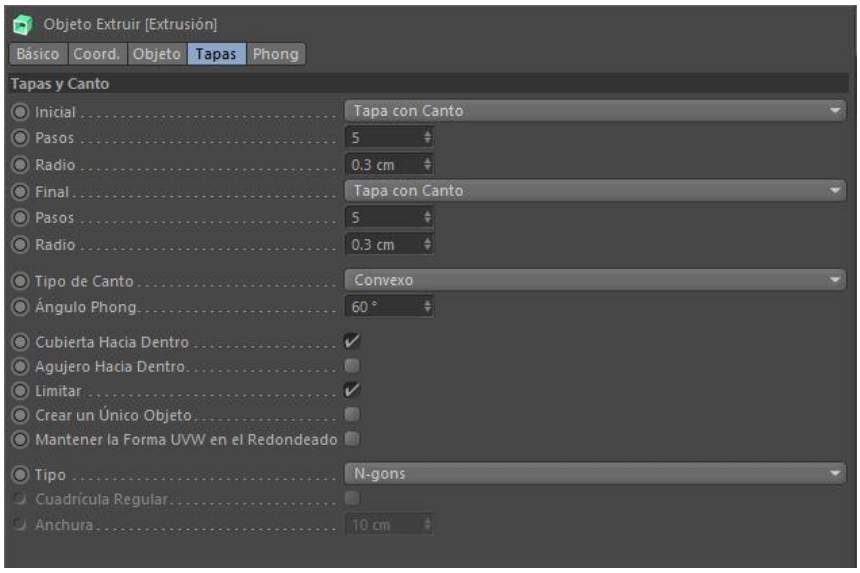

Figura 52. Adición de tapas laterales para redondear el filo y captar mejor la luz

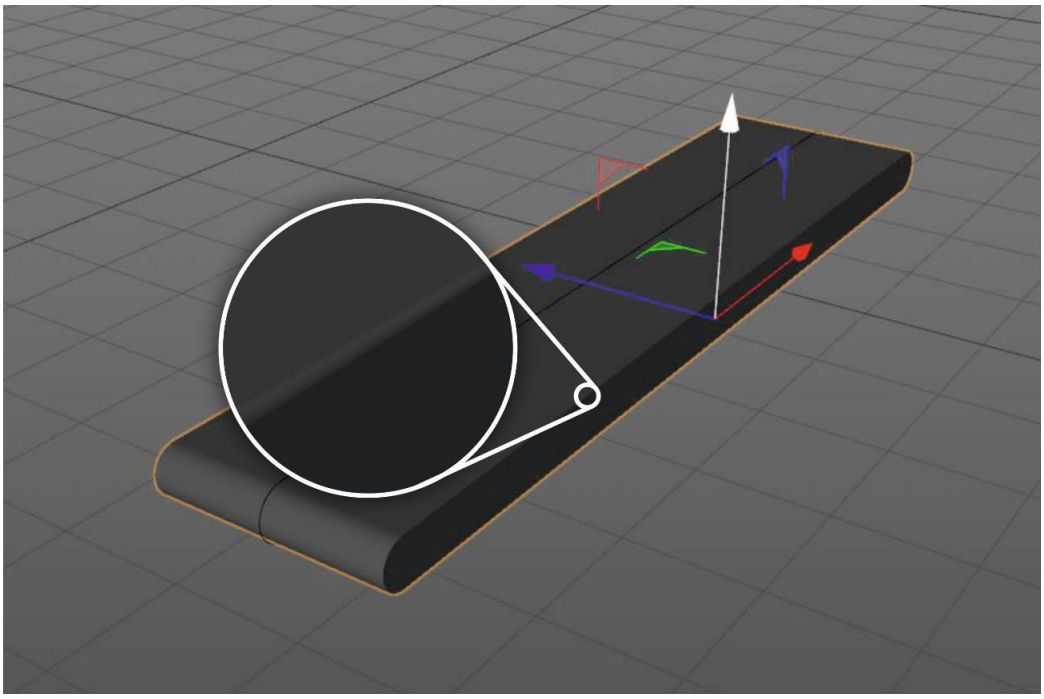

Figura 53. Detalle del canto redondeado del soporte central

6.1.2. Patas de la cinta

Para recrear las 4 patas que sostienen la cinta, se debe modelar por un lado la plataforma que se apoya en el suelo, y por otro lado la barra que une dicha plataforma con el soporte central.

El diseño de la plataforma es bastante sencillo. Basta con crear un círculo con las dimensiones deseadas, y añadirle una extrusión del tamaño del ancho de dicha plataforma.

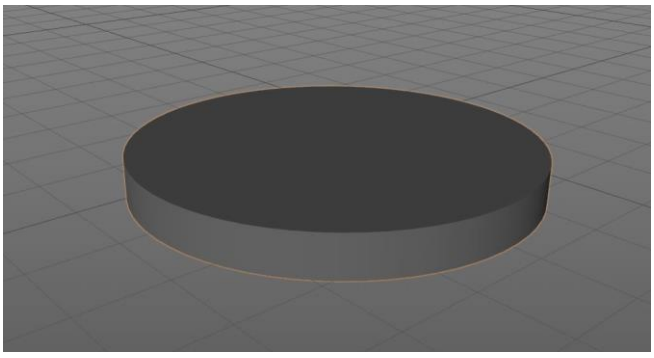

Figura 54. Plataforma de la pata de la cinta

El perfil de la barra se forma a partir de la combinación de un rectángulo y un círculo. Para unir ambas formas, se usa la herramienta *Unión Spline.* Una vez conseguido el perfil, bastan con aplicarle una extrusión del ancho deseado.

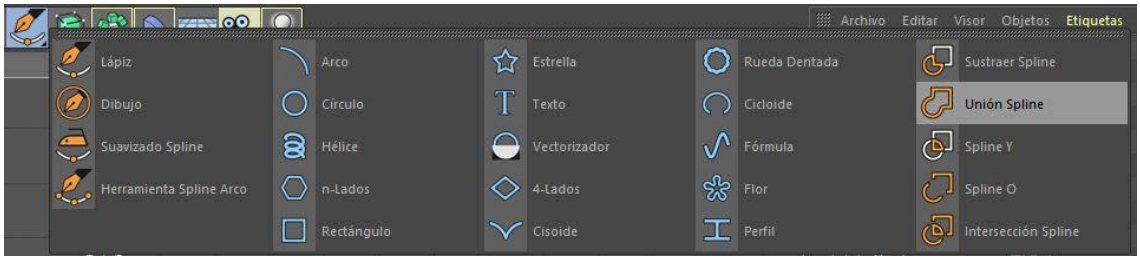

Figura 55. Función unión de splines en Cinema4D

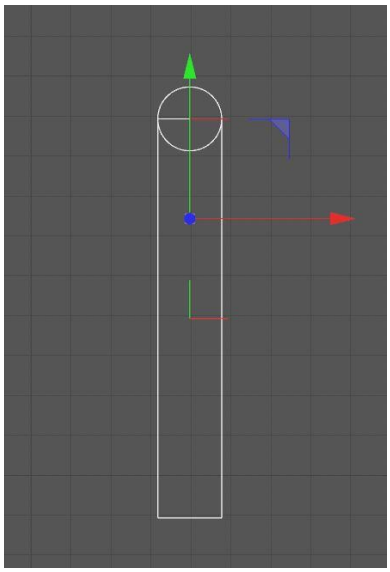

Figura 56. Perfil de la barra proveniente de la pata de la cinta

Para finalizar la pata, se unen las dos piezas formando un conjunto.

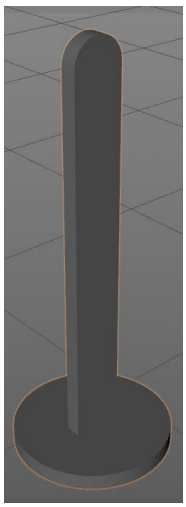

Figura 57. Unión de la plataforma y la barra para formar la pata completa de la cinta

6.1.3. Cinta de transporte

A continuación, se explica la creación de la cinta o cadena de transporte, la parte fundamental de este objeto, que realiza la acción de trasladar los objetos colocados encima de la misma. Para este elemento es necesario añadirle una animación, para simular el movimiento de la cinta, que no será tarea compleja en Unity, debido a la amplia variedad de herramientas que dispone.

En este caso, el diseño que se ha elegido es una especie de cadena, compuesta por numerosos rectángulos unidos que van girando. La filosofía de este modelado se basa en el diseño de un eslabón, y su clonación a lo largo de una guía con la forma de la trayectoria.

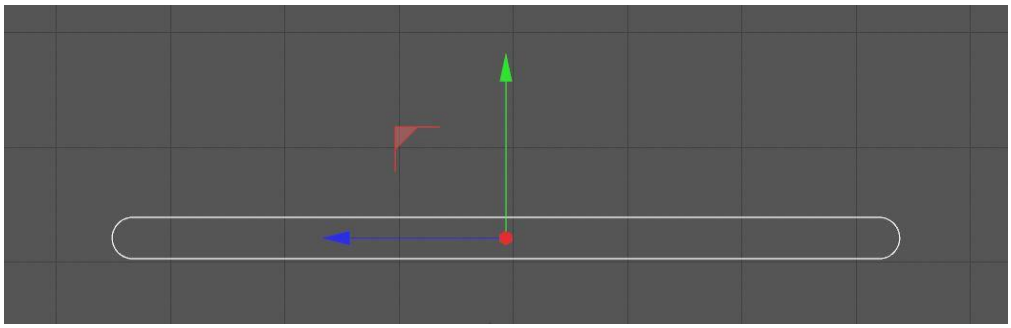

Figura 58. Perfil de la guía de la cadena

Por tanto, en primer lugar es necesario crear la guía. Como la cinta irá rotando alrededor del soporte central explicado anteriormente, la guía compartirá la forma del perfil de dicho soporte, aumentando sus proporciones 3cm a lo largo y ancho, resultando un perfil de 689 cm de ancho y 36 cm de alto.

Figura 59. Dimensiones y configuración de la guía

Una vez creada la guía, se procede a diseñar el primer bloque eslabón de la cadena o cinta, que se trata básicamente de un cubo al que se le modifican las dimensiones y redondean los cantos.

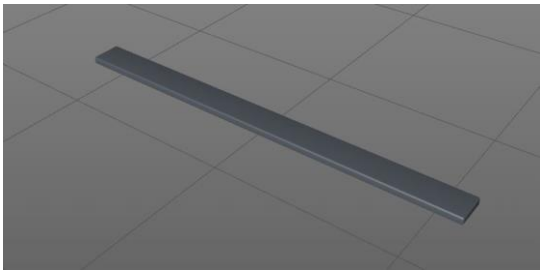

Figura 60. Bloque componente de la cadena de la cinta

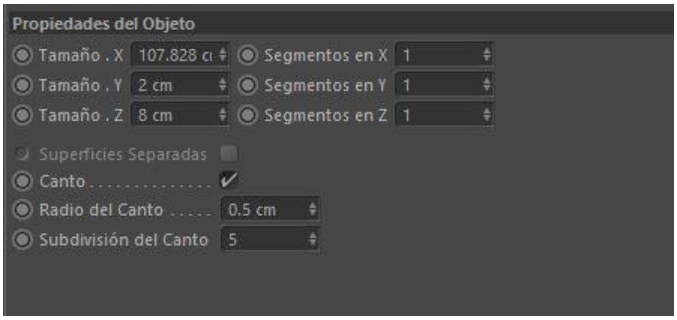

Figura 61. Dimensiones de cada bloque componente de la cadena

Para generar la cinta completa, se usa una herramienta llamada *Clonar,* que permite reproducir un elemento tantas veces como se fije a lo largo de una trayectoria, en este caso la guía creada anteriormente. Para usar esta herramienta, seleccionar *MoGraph > Clonador.*

Figura 62. Herramienta clonador de Cinema4D

Una vez creada la herramienta, se procede a su configuración. En primer lugar, se elige la configuración tipo *Objeto,* ya que el clonado se realizará siguiendo la traza del objeto *Rectángulo,* guía de la trayectoria. Además en el campo *Objeto* se selecciona nuestra guía.

El campo *cantidad* indica el número de objetos a crear.

Hasta aquí la configuración del diseño. Los últimos campos corresponden a la animación de giro de la cinta. Los campos inicio y fin indican los elementos de la clonación que corresponden al comienzo y fin de la animación. Para crear una cinta completa, debe seleccionarse el inicio en el 0% y el fin en el 100%, para cerrar el lazo de la cinta. De no ser así, habría un hueco en la misma.

El campo cíclico se activa para que la animación vuelva a comenzar una vez terminada.

Por último, la frecuencia gestiona la velocidad de la animación. En este caso se ha escogido un 6%.

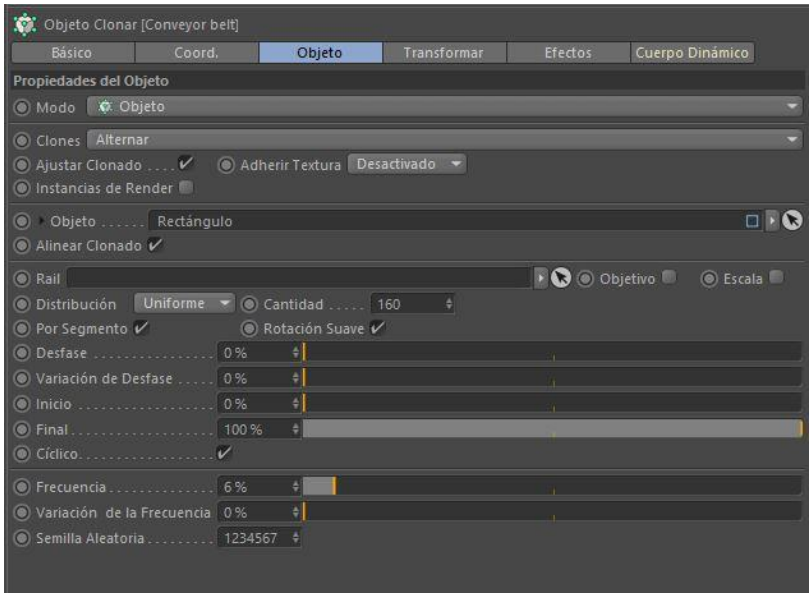

Figura 63. Configuración de la animación de la cinta

Para terminar, es necesario situar el objeto *Cubo* (eslabón de la cadena) dentro de la herramienta *Clonar*. Hecho esto, se generará la cinta completa.

Figura 64. Ubicación del cubo dentro del objeto clonador para aplicar dicha herramienta

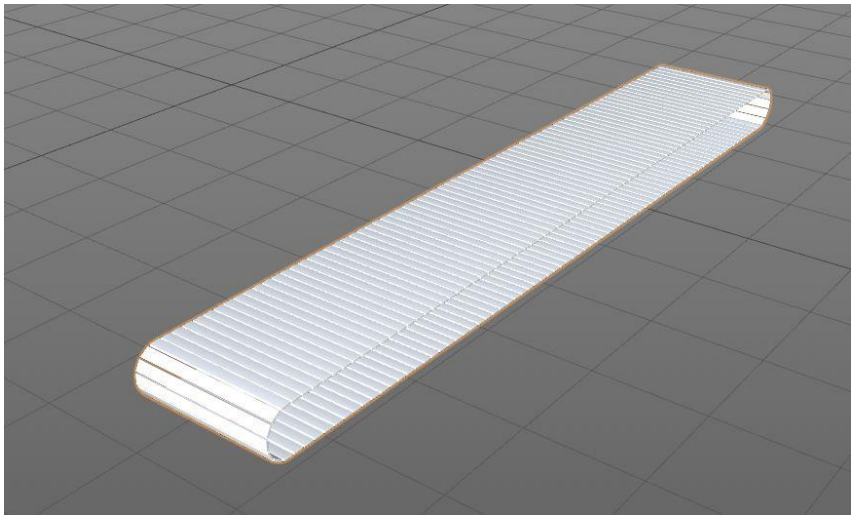

Figura 65. Vista general de la cadena de la cinta

Juntando todos los elementos, queda completado el diseño de la cinta transportadora.

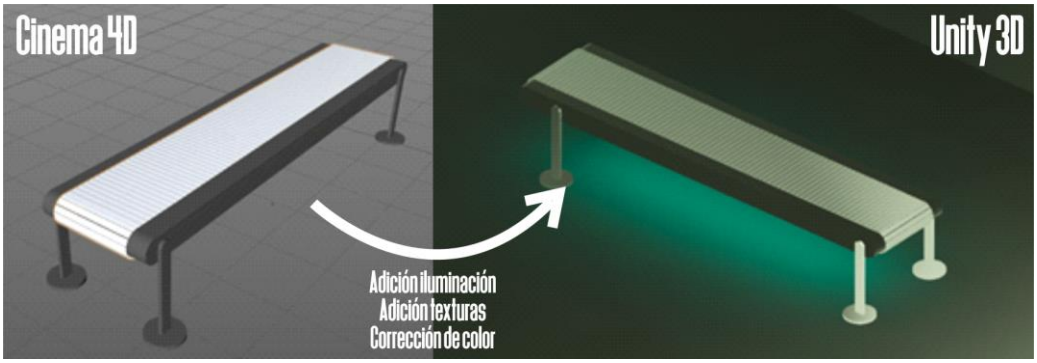

Figura 66. Comparación de la cinta creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color

6.2. Mesa giratoria

Este es uno de los elementos más complejos del proyecto en cuanto al diseño, ya que consta de varias partes móviles. Para la explicación, se hará por partes: la base, parte elevadora y cinta.

Figura 67. Apariencia final de la mesaq giratoria

6.2.1. Base

La base es la parte de la mesa que permanece estática en todo momento. Su modelado es bastante sencillo.

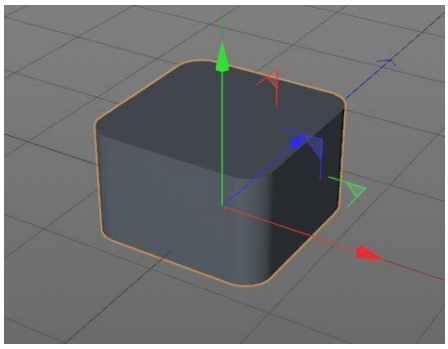

Figura 68. Pieza base de la mesa

El modelado de esta pieza se basa en el diseño del perfil, y la extrusión del mismo, para darle volumen, con la longitud deseada.

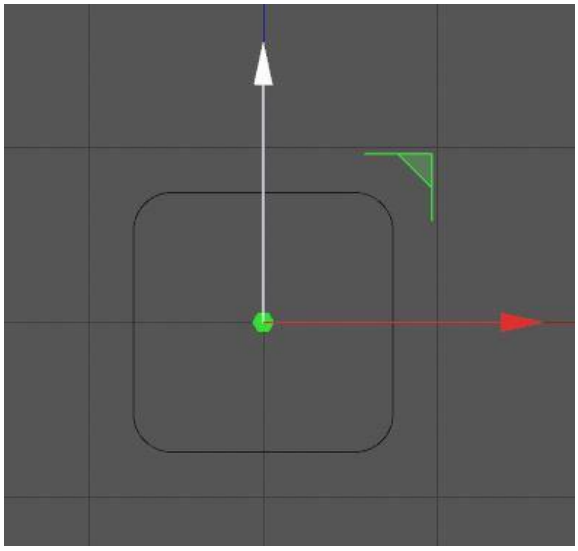

Figura 69. Perfil de la base de la mesa

El perfil del mismo se trata de un cuadrado de 149 cm de lado con las esquinas redondeadas. Este redondeo es una opción disponible, en la cual hay que definir el radio de giro de la curva.

Figura 70. Configuración de redondeo de esquinas del perfil de la base de la mesa

Una vez realizado, basta con aplicar una extrusión de 90.122 cm para conseguir el volumen.

6.2.2. Parte móvil

Este módulo comprende el diseño de la parte de la mesa que describe algún tipo de movimiento, tanto vertical como rotativo.

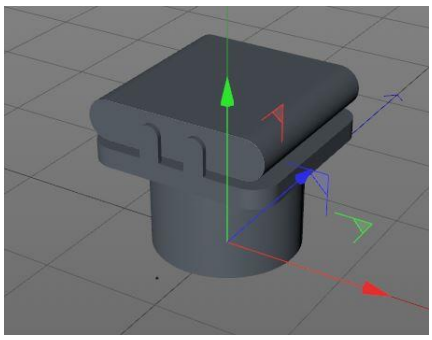

Figura 71. Parte móvil de la mesa

En primer lugar, se encuentra el cilindro de subida y bajada. Su diseño se basa en un cilindro de 59 cm de radio y 80 cm de alto.

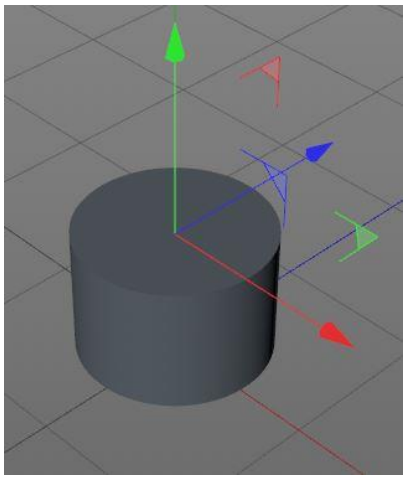

Figura 72. Cilindro elevador de la mesa

Apoyado sobre este se encuentra una plataforma base de los 4 soportes de la cinta. Esta plataforma comparte el perfil de la base. La diferencia se encuentra en el grosor, que en este caso son 16 cm. Cada uno de los soportes, de forma análoga, también comparten el perfil de las patas de la cinta transportadora, explicada anteriormente. Simplemente hay que modificar las dimensiones para que se adapten a su función.

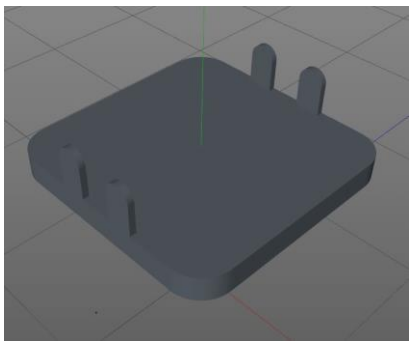

Figura 73. Plataforma base y cuatro soportes de la cinta

Por último se encuentra la pieza que realiza la función de guía sobre la cinta. Como las anteriores piezas, tiene una forma similar a la usada en el objeto de la cinta.

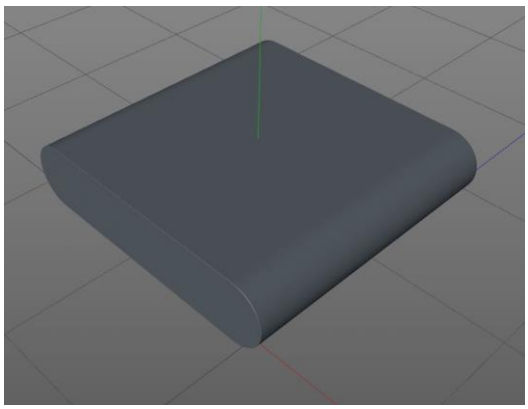

Figura 74. Soporte central de la cinta de la mesa giratoria

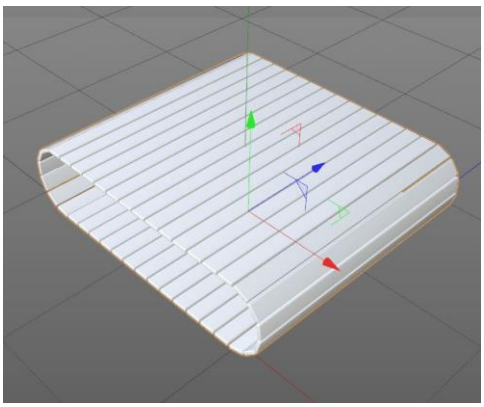

Figura 75. Cadena de la cinta de la mesa giratoria

Para concluir este objeto, es necesario añadir la cadena o cinta de transporte. Para ello, procediendo de la misma forma que en el objeto de loa cinta, generamos un eslabón y lo clonamos siguiendo el perfil de la guía. En este caso, como la cinta es más pequeña, para eslabones de las mismas dimensiones hará falta una cantidad inferior, concretamente 39.

Y de esta manera, uniendo todas las piezas se obtiene el modelado de la mesa giratoria.

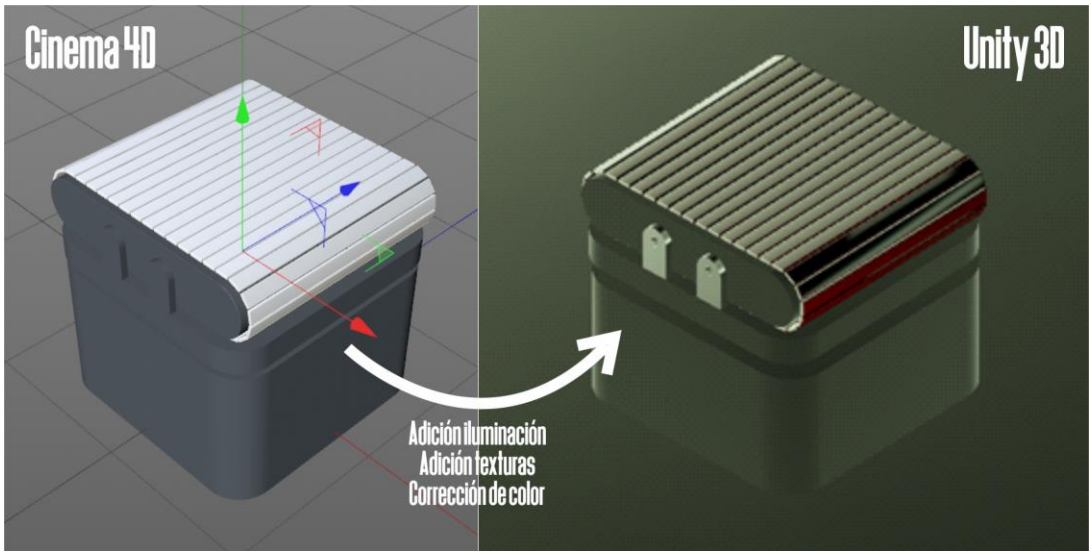

Figura 76. Comparación de la mesa giratoria creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color

6.3. Grúa pórtico

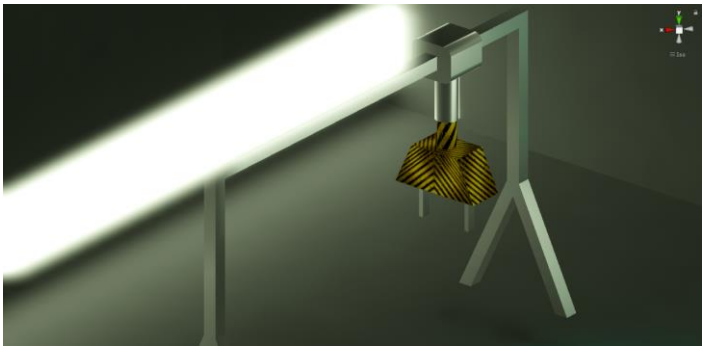

Figura 77. Apariencia final de la grúa

Este elemento consta de una parte fija, que conforma la estructura base, cuya función es la de sujetar la parte móvil, y además hace de guía para el movimiento trasversal. El resto de piezas constituye la parte móvil. A continuación, se analiza el modelado de cada una por separado.

6.3.1. Estructura base

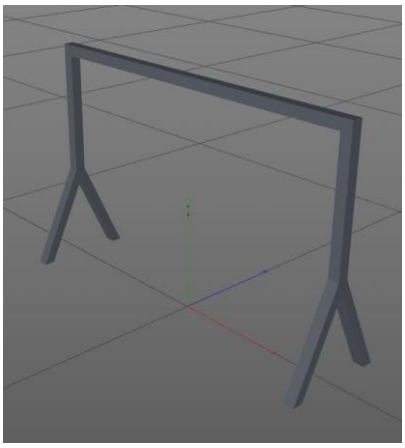

Figura 78. Apariencia final de la estructura base de la grúa

El diseño de esta estructura se ha realizado por partes. En primer lugar, se modela una de las patas, creando el perfil de la misma con la herramienta *Spline,* y se le añade volumen con *Extrusión.*

Figura 79. Perfil de una pata de la estructura (a) y aplicación de volumen al mismo (b)

En este caso, ´la herramienta extrusión y el perfil al que aplica se han unido en un solo objeto, más manejable, mediante la opción *Convertir a editable,* creando un nuevo objeto poligonal.

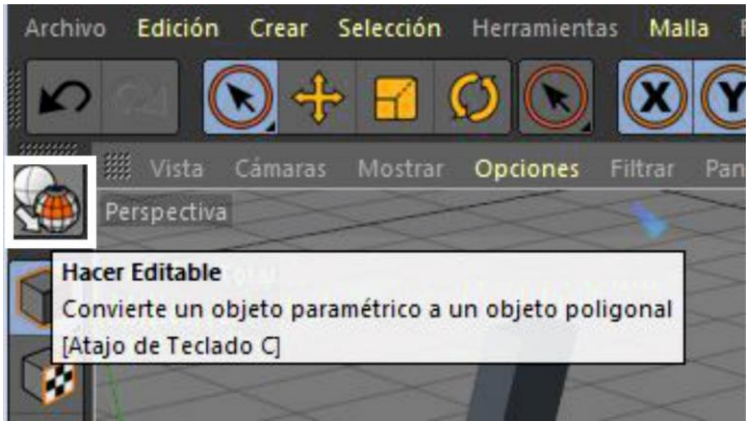

Figura 80. Herramienta Convertir a editable, para hacer el conjunto objeto-extrusión un único objeto más manejable

Una vez terminada una pata, es el momento de generar el resto a partir de ésta. Para ello se hace uso de la herramienta *Simetría,* la cual hace función de espejo, creando un objeto igual al seleccionado de forma simétrica dado un eje. Esta función se encuentra en la barra superior de herramientas.

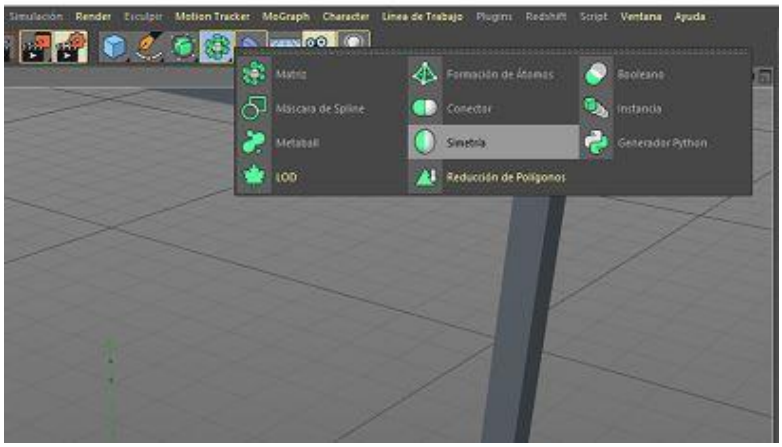

Figura 81. Herramienta Simetría de Cinema4D

Para aplicar la simetría a la pata, solo hay que ubicar esta debajo de la herramienta en la ventana de jerarquía, haciéndola *hija.*

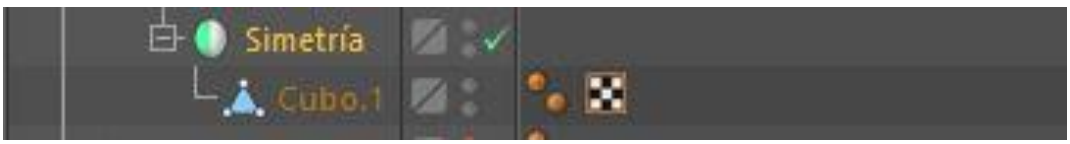

Figura 82. Colocación del objeto dentro del objeto Simetría para aplicar el efecto de la herramienta sobre el mismo

Además, es importante ubicar el eje de simetría correctamente para que se generen el resto de patas en la posición debida. Al necesitar cuatro patas, serán necesarias dos simetrías, las cuales se colocan ambas en el centro de la estructura, donde coinciden ambos ejes de simetría.

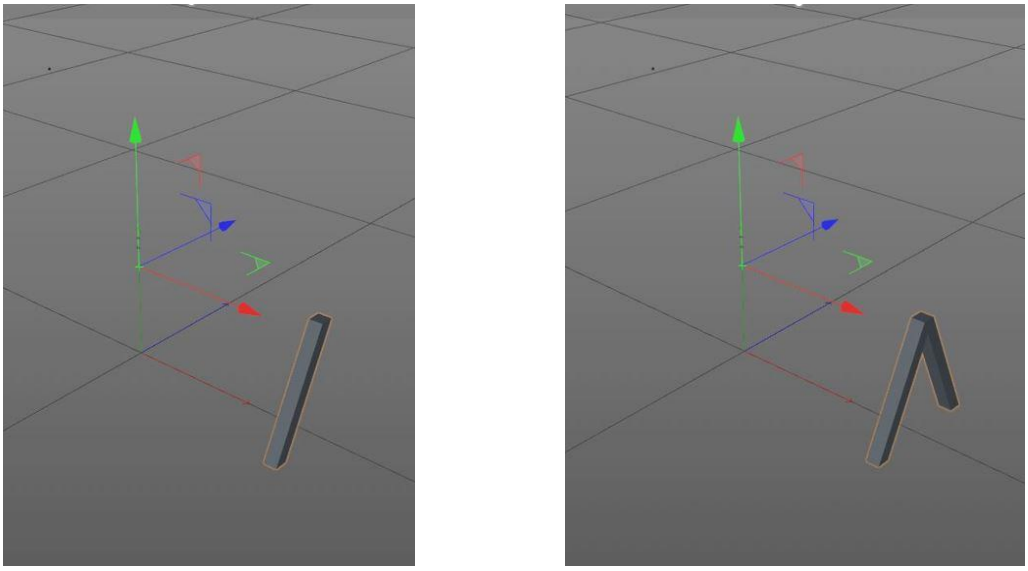

Figura 83. Duplicación de forma simétrica de una pata de la estructura de la grúa

Una vez duplicada la primera pata, se le añade el listón superior, y luego se realiza una segunda simetría, obteniendo las 4 patas. Para concluir la estructura, se crea la barra que une ambas patas, y que hará de guía para la pieza móvil.

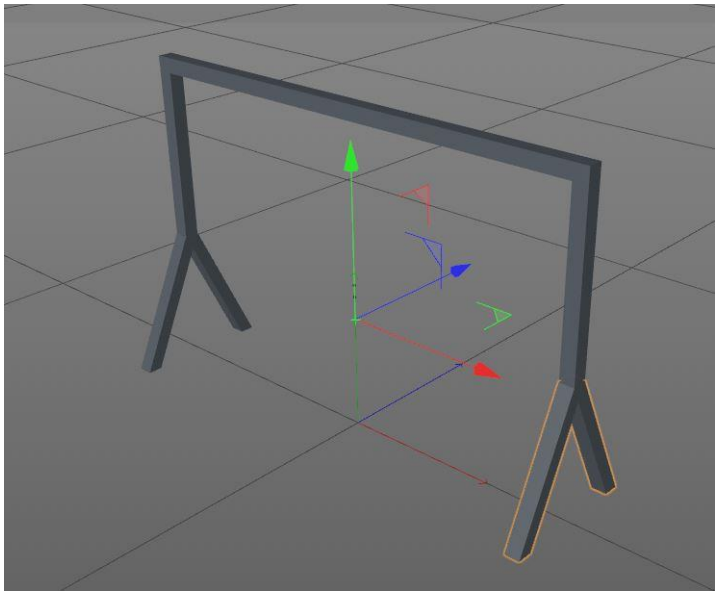

Figura 84. Estructura finalizada

6.3.2. Pieza móvil

Esta pieza es la encargada de mover la pinza de forma transversal a la estructura.

Su diseño se basa, como en otras piezas anteriores, en el diseño del perfil y su posterior extrusión.

El perfil de la pieza es básicamente un rectángulo con las esquinas redondeadas. Las dimensiones del rectángulo son 100 cm de ancho y 80 cm de alto, con un radio de redondeo de 20 cm. La extrusión aplicada posteriormente es de 100 cm en la dimensión perpendicular al plano del perfil.

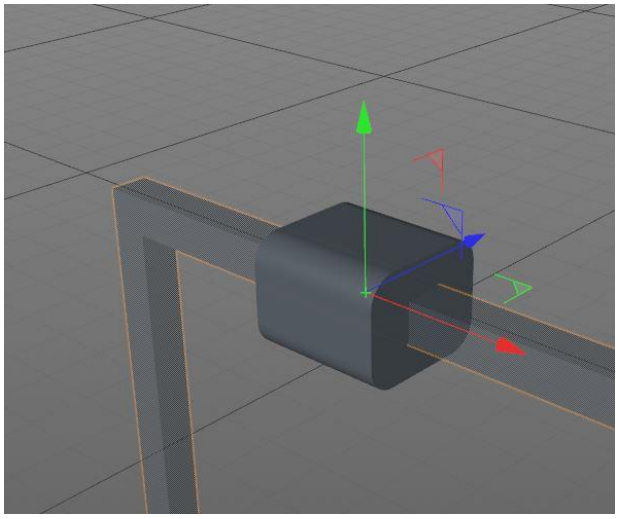

Figura 85. Pieza móvil de la grúa

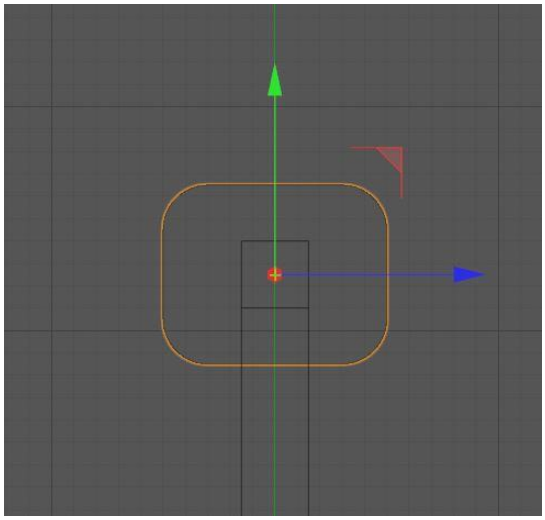

Figura 86. Perfil de la pieza móvil

6.3.3. Brazo extensible

Es el momento del brazo que conecta la pieza móvil con las pinzas, haciendo posible el desplazamiento, gracias a su composición de dos cilindros, uno de diámetro mayor al otro.

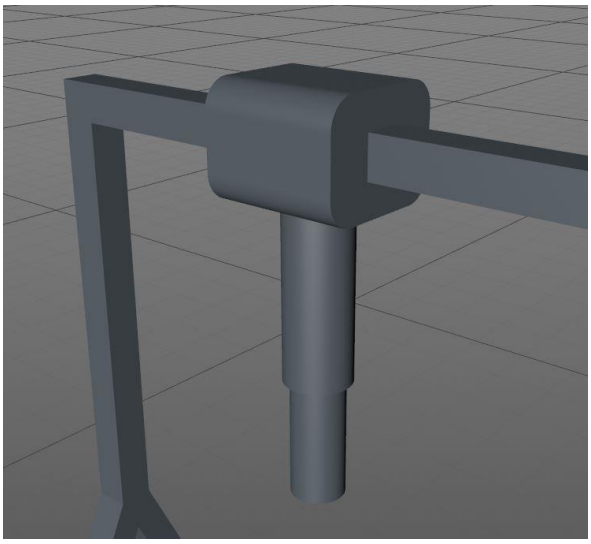

Figura 87. Brazo extensible de la grúa

Esta pieza se compone de dos cilindros, uno de ellos de mayor diámetro y hueco, para permitir que el otro pueda introducirse dentro de este. Concretamente, el cilindro de menos diámetro es de 40 cm. Su creación se basa en insertar un nuevo cilindro a través de la barra de herramientas, y definir el radio del mismo, así como su longitud. El otro, al ser hueco, tiene dos diámetros.

Figura 88. Dimensiones del anillo que forma el perfil del cilindro exterior

El menor de ellos coincide con el otro cilindro, 40cm. El radio grande es de 50 cm. Para la creación de este cilindro hueco se inserta un círculo con configuración de anillo (dos diámetros), y posteriormente se le aplica una extrusión de la longitud deseada.

El cilindro macizo tiene una longitud de 221.96 cm, mientras que la del hueco es de 150 cm.

6.3.4. Soporte y pinzas

Este conjunto de piezas completa el elemento de la grúa. Es el encargado de agarrar los objetos para transportarlos, por lo que es de crucial importancia.

En primer lugar se explica la pieza que mueve las pinzas, llamada a partir de ahora como *soporte.*

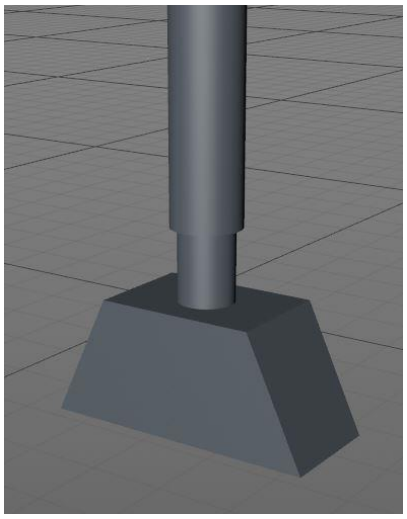

Figura 89. Pieza soporte de pinzas de la grúa

El desarrollo de este objeto se basa en el diseño de la mitad del perfil, la generación de su volumen mediante la herramienta extrusión, y finalmente la duplicación simétrica para obtener la pieza completa.

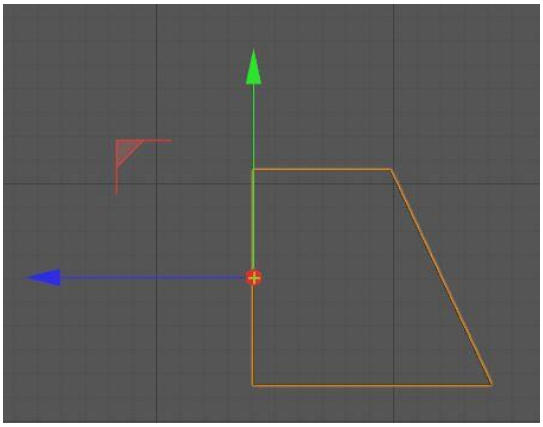

Figura 90. Diseño de la mitad del perfil de la pieza

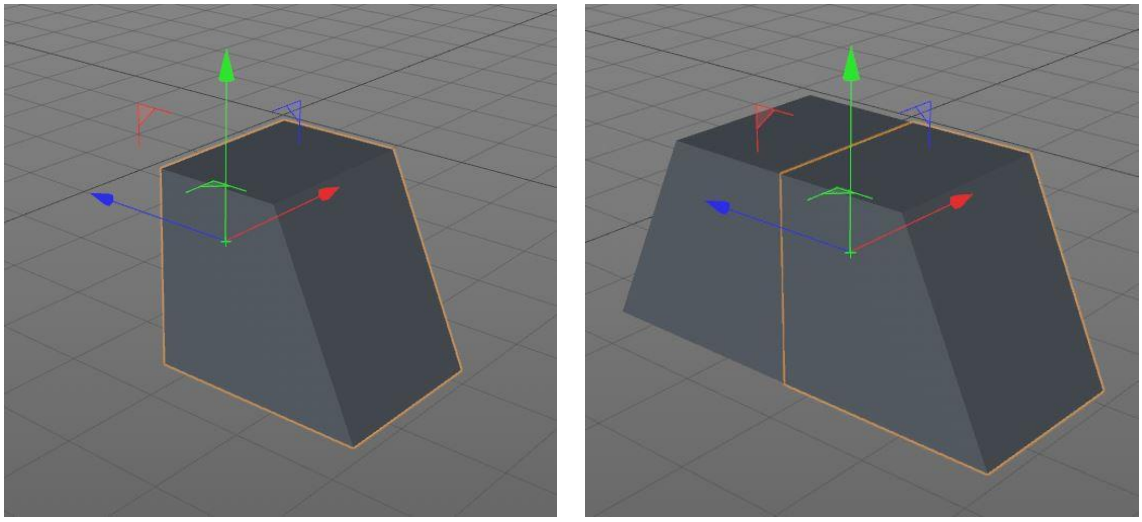

Figura 91. Obtención de la pieza completa mediante el uso de la herramienta Simetría

El diseño para las pinzas que se ha usado en este proyecto ha sido muy simple, basándose en dos barras rectas de tamaño apropiado a las cajas que agarra la grúa. Para ello, simplemente es necesario crear un cubo y determinar las dimensiones para que se ajuste a las condiciones de su funcionamiento. Inicialmente, cada pinza tenía unas dimensiones de 5 x 80 x 5 cm, pero luego, al exportar a Unity y comparar con las cajas, se notaron algo pequeñas, por lo que se procedió a agrandarlas hasta que su tamaño fuera adecuado.

6.3.5. Animaciones

Como se estudió previamente en la programación de la grúa, este elemento realiza tres tipos de movimiento diferentes: movimiento trasversal, vertical y apertura y cierre de las pinzas. De estos tres desplazamientos, tanto el desplazamiento trasversal como el vertical, a diferencia de la acción de las pinzas, hacían uso del Animator Controller, que era una forma más sencilla de generar un conjunto de movimientos mediante un diagrama de flujo y variables de activación de estados. De todos los estados, algunos de ellos, los estados de movimiento, tenían asociada la animación del objeto. Dicha animación se crea en Cinema 4D, y posteriormente es exportada con el objeto a Unity 3D.

Para la grúa se han generado dos pares de animaciones: movimiento trasversal de ida y vuelta, y movimiento vertical de ida y vuelta.

Como el procedimiento para la generación del movimiento vertical y el trasversal es el mismo, se explicará solo uno de los dos, por ejemplo el movimiento trasversal.

La forma de generar animaciones en Cinema 4D es muy sencilla. Basta con crear fotogramas claves con las posiciones de inicio y fin de la transición. Un fotograma clave es aquel que almacena la posición y orientación de los objetos activos. Si se crean dos fotogramas clave de un objeto en dos posiciones distintas, al reproducir la secuencia se generará un desplazamiento progresivo de una posición a otra.

Para la creación del movimiento transversal de la pieza móvil de ida y vuelta, se van a definir 4 posiciones clave, que son la de inicio y fin del movimiento de ida y de vuelta. Como es lógico, la posición de inicio del movimiento en un sentido coincide con la posición final del otro.

Figura 92. Línea de tiempo de Cinema4D

Para hablar de las animaciones, hay que estar familiarizado con la línea de tiempo, que indica la duración total de la animación, así como la posición de cada uno de los fotogramas clave.

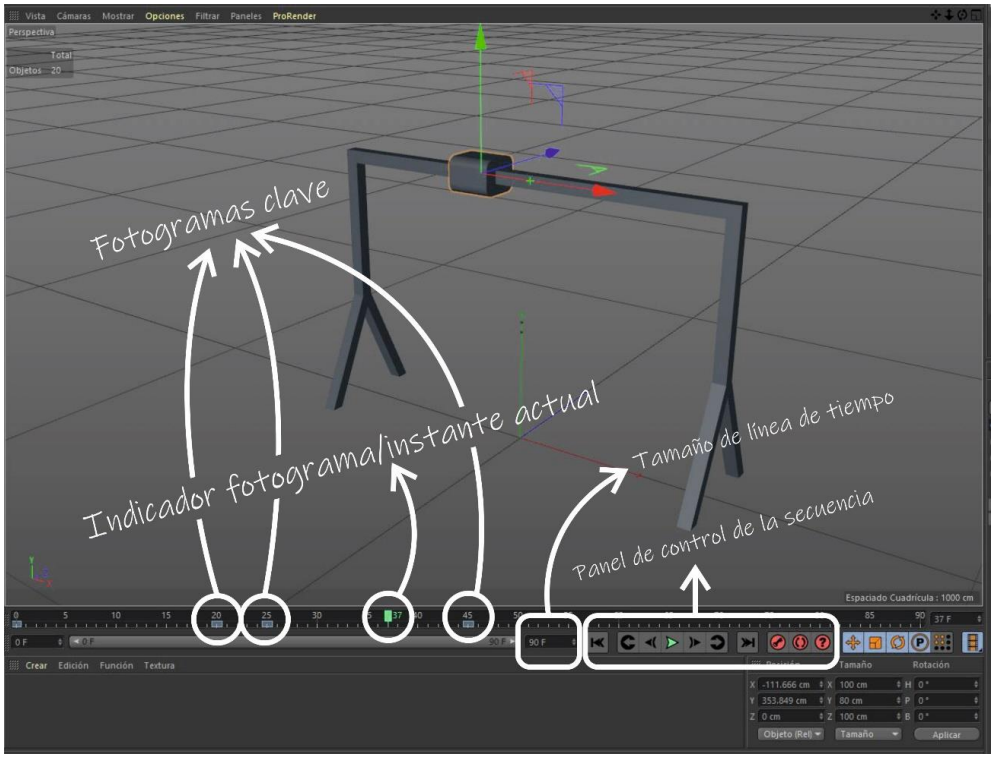

Figura 93. Interfaz de barra de herramientras de la seciencia

Además, el indicador verde va indicando el fotograma actual durante la reproducción de la animación. El tamaño de la línea de tiempo puede modificarse en el indicador que se encuentra debajo. Por otro lado, también puede cambiarse la unidad representada, por ejemplo a segundos, accediendo a los ajustes del proyecto.

A la derecha del indicador del tamaño de la línea de tiempo se encuentra un panel de control, donde, por un lado, se encuentran los botones para reproducir/pausar la animación, avanzar o retroceder la secuencia, desplazarse de un fotograma clave a otro, o incluso al primer o último fotograma. Por otro lado, a su derecha están los botones dedicados a los fotogramas clave. En este caso, el que se va a usar es el primero, cuya función es la creación un fotograma clave del objeto seleccionado en la posición en la que se encuentre el indicador de la línea de tiempo.

Por tanto, el procedimiento es el siguiente:

- Mover el indicador de la línea de tiempo al primer fotograma.
- Seleccionar el objeto al que aplica la animación. En este caso se trata de la pieza móvil.
- Ubicarlo en la posición de comienzo de la animación.
- Generar el primer fotograma clave pulsando el botón .
- Desplazar el indicador de la línea de tiempo al final de la animación deseado.
- Ubicar el objeto en la posición final de la animación.
- Generar el segundo fotograma clave con el botón .

Una vez realizado todo el procedimiento, se puede observar que, al reproducir la secuencia, el objeto se desplazará de un punto a otro con la duración concretada entre fotogramas claves, que en este caso es de 10 fotogramas.

Para crear el movimiento inverso, se dejan 5 fotogramas de espacio, y se vuelve a crear un fotograma clave en la posición final. Luego, con una separación de otros 10 fotogramas, ubicando el objeto en la posición inicial, se genera el cuarto y último fotograma clave, obteniendo la animación de ida y vuelta.

Esta explicación aplicada al movimiento trasversal de la grúa es aplicable a cualquier otra animación, ya que el procedimiento es exactamente el mismo.

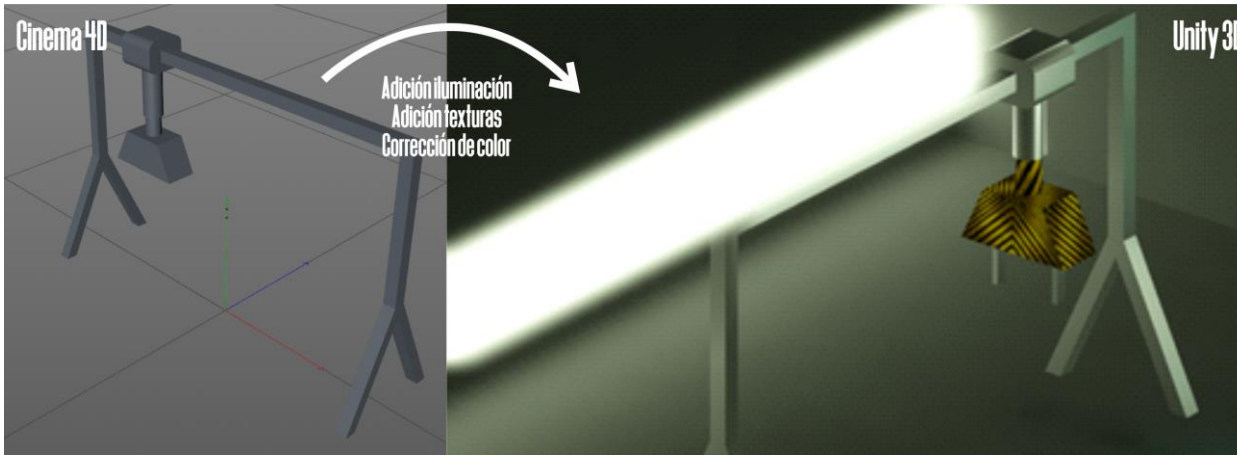

Figura 94. Comparación de la grúa creada en Cinema4D y la misma importada en Unity tras aplicarle iluminación, texturas y corrección de color

6.4. Pistón

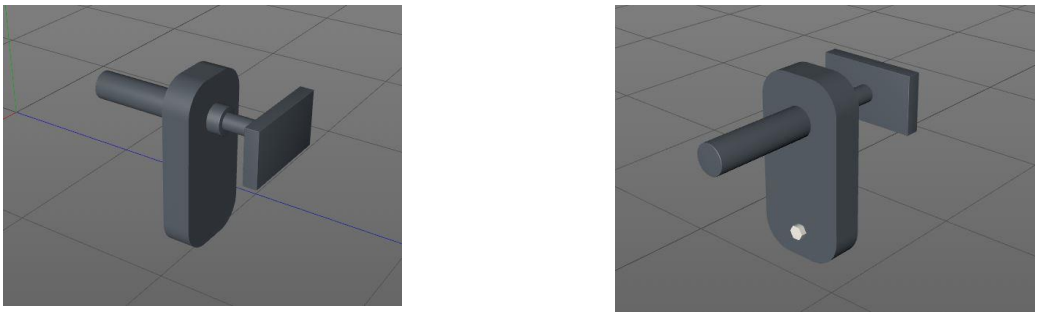

Figura 95. Vista general del piston de encuadre

El pistón es el elemento de encuadre de cajas para que se orienten correctamente y la grúa pueda agarrarlas.

En cuanto al diseño, se ha creado un cilindro fijo hueco del cual sale otro de menor diámetro que tiene acoplada una superficie pensada para empujar las cajas. Esta estructura la soporta una placa rectangular con las esquinas redondeadas y sujetada por un tornillo de cabeza hexagonal. A continuación se explica el modelado de cada pieza.

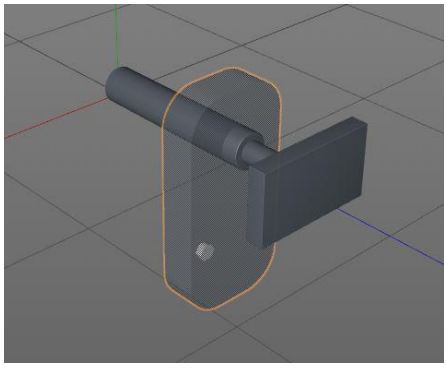

Figura 96. Parte móvil del pistón

Para comenzar, la "parte dinámica" se compone de dos cilindros. El mayor tiene un diámetro de 10 cm y longitud de 50 cm. El pequeño, que se ubica dentro del grande, tiene un diámetro de 6 cm y longitud de 67 cm. Es importante que este sea más largo que el grande, para permitir un mayor recorrido de empuje. Acoplado a éste se encuentra la placa, que básicamente se trata de un cubo con dimensiones 35 x 20 x 5 cm, creado como "hijo" del cilindro móvil, para que ambos elementos se desplacen de manera unitaria.

Por otro lado, el diseño de la pieza soporte es bastante sencillo. En primer lugar, se define el perfil de la misma, como un rectángulo de 61 x 30 cm, con las esquinas redondeadas con un radio de 10 cm. Luego, simplemente es necesario aplicarle volumen con la herramienta "*Extrusión*", con un ancho de 10 cm, teniendo la pieza soporte completada.

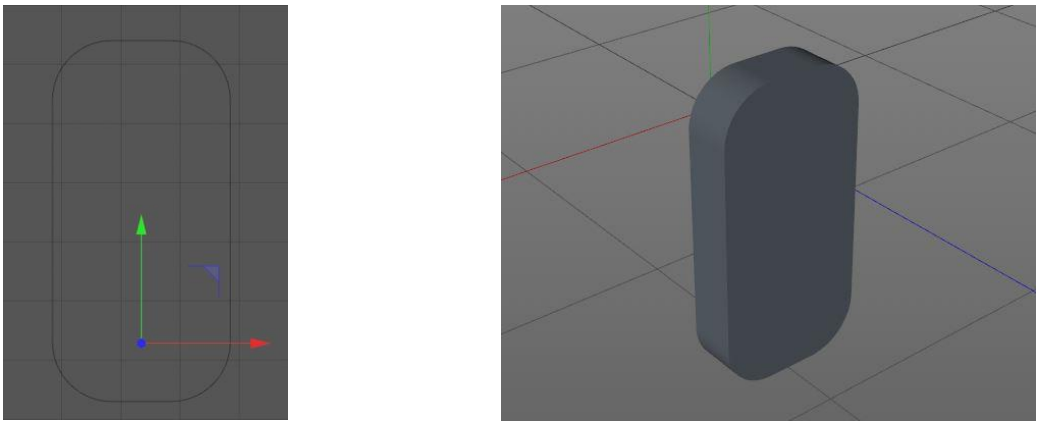

Figura 97. Perfil del soporte y soporte con volumen aplicado

Finalmente se encuentra el tornillo, del cual solo es necesario diseñar su parte superior, que es la que se aprecia al estar enroscado en la pieza soporte. Este diseño es similar al anterior. El perfil del tornillo es un hexágono. Esta forma se genera en Cinema 4D insertando un objeto denominado "n-Lados", situado en el panel superior, definiendo sus parámetros, es decir, un radio de 2.5 cm, 6 lados, y situado en el plano XY. Creado el perfil, se le añade el volumen con "*Extrusión*" de 2 cm.

Y así finalizaría el diseño del pistón.

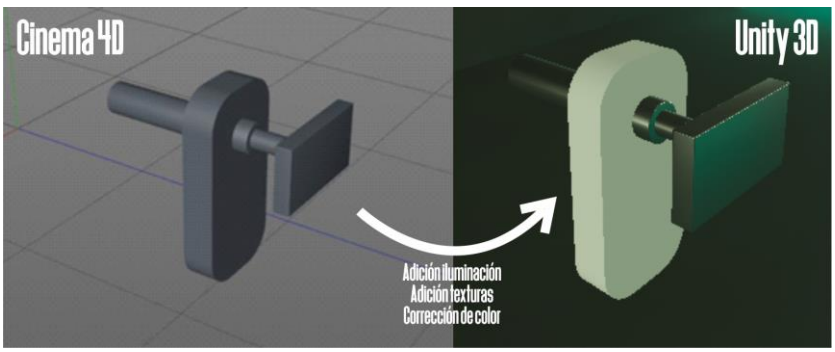

Figura 98. Comparación del pistón creado en Cinema4D y el mismo importado en Unity tras aplicarle iluminación, texturas y corrección de color

6.5. Sensor láser

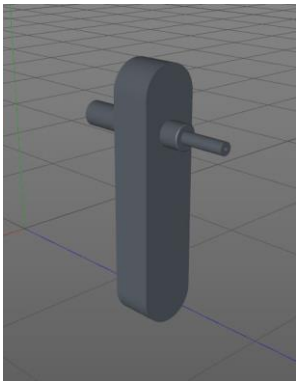

Figura 99. Vista general del sensor láser

Este elemento es muy similar al pistón, por lo que se hará un resumen de su diseño.

Los dos cilindros tienen diámetros de 3 y 6 cm. El cilindro de menor diámetro contiene un hueco de 1 cm de diámetro. Para generarlo, basta con crear un círculo con configuración de anillo, y aplicarle posteriormente una extrusión.

La pieza que hace las funciones de soporte tiene un perfil generado a partir de un rectángulo de 61 x 14 cm, con las esquinas redondeadas con un radio de 7 cm, y una extrusión de 10 cm, igual que la pieza del pistón.

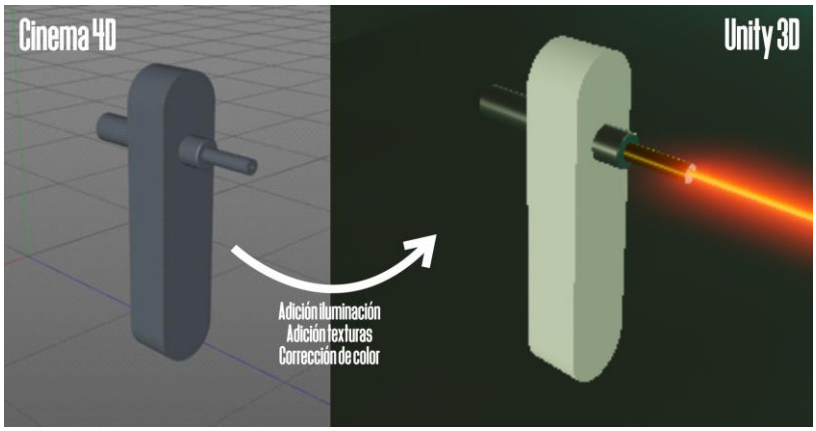

Figura 100. Comparación del sensor láser creado en Cinema4D y el mismo importado en Unity tras aplicarle iluminación, texturas y corrección de color

7 DESCRIPCIÓN DE LA PLANTA INDUSTRIAL DISEÑADA

Una vez explicada la programación y el comportamiento de todos y cada uno de los elementos funcionales, a continuación se expondrá una visión global de la planta, comentando todos los elementos que contiene y la conexión entre ellos.

La planta industrial propuesta en este proyecto realiza la función de transportar cajas de un punto a otro aplicando dos rotaciones y por medio de cintas y una grúa. Estas características pueden ser apropiadas para ciertas especificaciones. Poniendo un ejemplo, la traslación por medio de la grúa puede resultar útil para colocar la caja encima de un pallet ubicado en la posición final de la grúa, cosa que no sería posible con una traslación por medio de una cinta. Por otro lado, las rotaciones de las mesas permiten que la caja lleve siempre la misma orientación respecto a la dirección del movimiento. Esto permitiría colocar sensores en los laterales de la caja que leyeran alguna etiqueta digital colocado en el lateral de la caja, y que fueran validando los puestos por los que va pasando dicha caja.

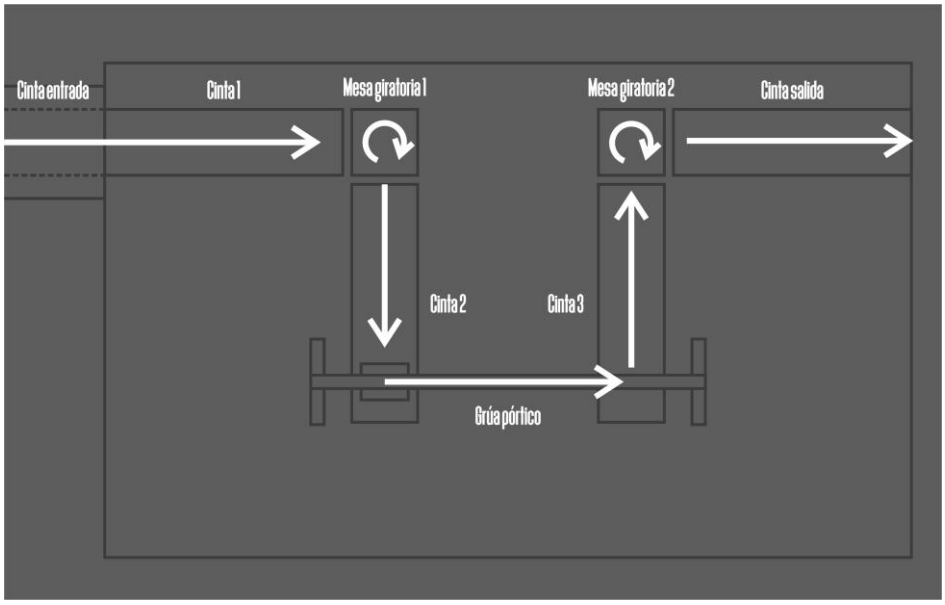

Figura 101. Esquema general del flujo de movimiento de la planta

7.1. Resumen de elementos de la planta

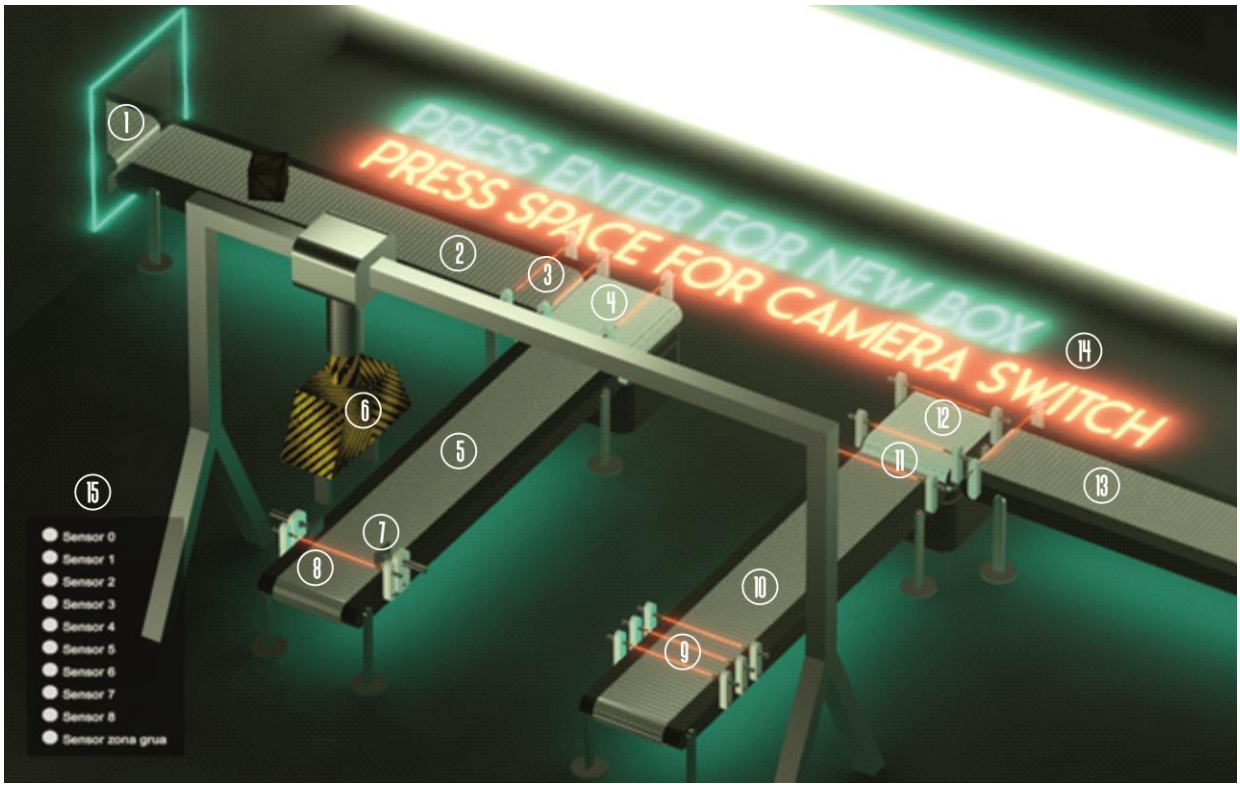

Figura 102. Vista general de la planta con corrección de color aplicada

En la figura 102 se observa una vista global de la planta con todos sus elementos enumerados, mencionados a continuación:

- **Cintas transportadoras:** una de entrada de cajas (1), otras tres cintas intermedias (2, 5 y 10), y una cinta de salida (13).
- **Sensores de posición (3, 8, 9 y 11):** colocados a lo largo de todo el recorrido para conocer la posición de la caja en todo momento. Están ubicados al principio y al final de cada elemento para conocer el momento de activación de cada
- **Mesas giratorias:** se han colocado dos en esta planta, una de ellas antes del traslado de la grúa (4) y otra después (12).
- **Pistones (7):** ubicados justo antes del agarre de la grúa para corregir la orientación de la caja.
- **Grúa portico (6):** encargada de transportar las cajas desde la cinta (5) a la cinta (10).
- **Iluminación y letreros de neón (14):** elementos encargados de la iluminación global de la planta. Además se han instalado placas lumínicas de color azul debajo de cada cinta transportadora, a modo decorativo.
- **Panel de información de sensores (15):** colocado en la parte inferior izquierda de la pantalla, muestra el estado de cada uno de los sensores colocados en la planta.

7.2. Coordinación de todos los elementos. Explicación del script *Control.*

Para hacer possible la coordinación de todos los elementos presents en la planta, se ha generado el script *Control*. Este programa se encarga de activar y desactivar el funcionamiento de cada uno de los elementos presents en base a la posición de la caja, ubicada gracias a la información de los sensores.

La estructura del programa es la que sigue:

1. Declaración de todas las variables necesarias
2. Inicio del bucle. El código del bucle se ejecuta de forma cíclica.
    a. Inicialización de las variables a partir de los valores de las variables de los elementos asociados. En este momento, el programa recopila información de todos los sensores de la planta y los almacena en variables.
    b. Actuación sobre los elementos respecto a la información de los sensores. En este punto se describe el comportamiento de cada uno de los elementos (cintas, mesas, grúa, pistones, generador de cajas). Para este apartado, se ha usado una estructura de programación denominada *switch case*, la cual permite definir valores a las variables de salida (actuadores) dependiendo del valor de las señales de entrada (sensores). A continuación puede apreciarse, a modo de ejemplo, la descripción del comportamiento de la cinta 2.

En este caso concreto, los sensores que intervienen son el sensor 4 y los sensores de apertura y cierre del pistón. Los actuadores son el interruptor de la cinta 2 y el interruptor del pistón.

```
switch (estadocinta2)
{
    case 0:

        flagcinta2 = true;
        flagpiston = false;
        if (sensor4)
        {
            estadocinta2 = 1;
        }
        break;

    case 1:
        flagcinta2 = false;
        flagpiston = true;
        if (pistoncerrado)
        {
            estadocinta2 = 2;
        }
        break;
    case 2:
        flagcinta2 = false;
        flagpiston = false;

        if (pistonabierto)
        {
            estadocinta2 = 3;
        }
        break;
    case 3:
        flagcinta2 = false;
        flagpiston = false;

       if (estadogrua == 4 && !sensor4)
        {
            estadocinta2 = 0;
        }

        break;
```

8 CONCLUSIONES Y TRABAJOS FUTUROS

Tras la explicación de todos los ámbitos de este proyecto, a continuación se hace un resumen de todos los objetivos conseguidos, y posteriormente se plantean ciertas ideas para implementar en el futuro, enriqueciendo aún más el simulador desarrollado.

Dentro de todas las ideas que han podido llevarse a cabo en este proyecto, destacan las siguientes

- Diseño de varios componentes propios de una planta industrial (cinta transportadora, mesa giratoria, grúa pórtico…) tanto de forma visual como física, simulando el comportamiento real de cada elemento.
- Coordinación y automatización de todos los elementos de la planta, consiguiendo transportar una caja desde el comienzo de la planta hasta el final pasando por todos los elementos desarrollados sin necesidad de ninguna acción por parte del usuario más que el pulsado de la tecla *intro* para crear nuevas cajas que lleguen a la planta.
- Generación de una segunda cámara de vista en primera persona para que el usuario pueda desplazarse y orientarse por el interior de la planta usando las teclas *A, S, D* y *W* y el puntero del *mouse,* cambiando de cámara con la barra espaciadora.
- Implementación de panel visual con la información del estado de cada uno de los sensores colocados en la planta, para verificar su correcto funcionamiento.
- Adición de un sistema de iluminación realista, con placas luminiscentes, luces de neón y células de almacenamiento de información lumínica, así como corrección de color para mejorar la experiencia visual y conseguir una apariencia más estética y realista.

Por otro lado, se exponen a continuación posibles aportaciones futuras al proyecto, ideas que complementarían y añadirían nuevas funcionalidades y ventajas al proyecto. Algunas de esas ideas son las siguientes:

- *Diseño de plataforma hardware para conexión con cualquier controlador por un protocolo establecido. Esto haría posible usar dicho simulador en aplicaciones reales de la industria.*
- *La creación de un editor de escenarios que permita añadir, quitar y distribuir elementos en una planta conforme decida el usuario. Esta implementación haría del simulador una herramienta mucho más versátil para el usuario, ya que podría crear una planta que cumpliese las especificaciones requeridas por el usuario*
- *Desarrollar una estructura de programación de objetos estándar. Esta idea haría posible el hecho de que el usuario generara nuevos objetos que se añadieran a la biblioteca, y así poder ser usados por otros usuarios. Dicha estructura tendría que estudiarse, aunque tendría una cabecera de este tipo:*

  Class objeto (vector x, vector y, vector actuaciones digitales, vector actuaciones analógicas, vector llamadas_métodos, posición inicial, velocidad, struct [])

9 BIBLIOGRAFÍA

[1] González Muñoz C., Gracia Bandrés, M. A., Sanagustín Grasa, L., Romero San Martín, D., 2015. Análisis Motores gráficos y su aplicación en la industria. TecsMedia. Instituto Tecnológico de Aragón, ITAINNOVA."
[2] Motor de físicas, 2013. GamerDic, Diccionario online de términos sobre videojuegos y cultura gamer. URL. http://www.gamerdic.es/termino/motor-de-fisicas. Fecha de consulta: 29 de Junio de 2020.

[3] Adolfo J. Sánchez, J. M. Escaño and C. Bordons, "Simulator for control and automation using an interactive and configurable 3D virtual environment" 2012 Proceedings of SICE Annual Conference (SICE), Akita, 2012, pp. 2268-2273.
[4] Álvaro Rivero Portillo, 2018. "Desarrollo de plantas virtuales mediante Unity 3D. Interconexión Modbus/TCP-IP con PLC industrial. Trabajo Fin de Grado. Universidad Loyola Andalucía.
[5] Unity Documentation. Descargando e Instalando Unity, 2016. URL: https://docs.unity3d.com/es/530/Manual/InstallingUnity.html. Fecha de consulta: 29 de Junio de 2020.
[6] José Antonio González Seco, 2001. El lenguaje de programación C#. URL: https://programacion.net/articulo/el_lenguaje_de_programacion_c_167/3. Fecha de consulta: 29 de Junio de 2020.
[7] KEYENCE Corporation, 2020. Fundamentos del sensor. ¿Qué es un sensor fotoeléctrico?. URL: https://www.keyence.com.mx/ss/products/sensor/sensorbasics/photoelectric/info/. Fecha de consulta: 29 de Junio de 2020.
[8] Josué David Moreno, 2019. ¿Qué es Cinema 4D?
URL: https://www.calamoycran.com/blog/que-es-cinema-4d. Fecha de consulta: 29 de Junio de 2020.